(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 803 826 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
29.10.1997 Bulletin 1997/44

(51) Int Cl.[6]: **G06F 17/30**, H04N 7/173

(21) Application number: **97302676.8**

(22) Date of filing: **18.04.1997**

(84) Designated Contracting States:
DE FR GB IT NL SE

(30) Priority: **22.04.1996 US 636118**

(71) Applicant: **SUN MICROSYSTEMS, INC.**
Mountain View, CA 94043 (US)

(72) Inventors:
- **Lindblad, Christopher**
  Stanford, California 94309 (US)
- **Cachat, Stephan E.**
  Mountain View, California 94041 (US)

(74) Representative: **W.P. Thompson & Co.**
Coopers Building,
Church Street
Liverpool L1 3AB (GB)

(54) Video on demand applet method and apparatus for inclusion of motion video in multimedia documents

(57) The present specification describes a computer process which requests streams of motion video titles and decodes and displays the motion video signals of the stream for display in a computer display device is constructed in the form of an applet 212 of a multimedia document viewer 202 such as a World Wide Web browser. Accordingly, a designer of multimedia documents such as HTML pages can easily incorporate motion video titles into such HTML pages by specifying a few parameters of a desired title or a desired portion of a title to be requested from a video server 250. The applet 212 builds bit stream control signals from the specification of the title or the portion of the title. The bit stream control signals request transmission of the title or the portion of the title from a bit stream server such as a video server 250 and are in a form appropriate for processing by the bit stream server. The applet 212 transmits the bit stream control signals to the bit stream server 250 to thereby request that the bit stream server 250 initiate transmission of a bit stream representing the requested title or the requested portion of the title. The applet 212 also builds decoder control signals from the specification of the title or the portion of the title. The decoder control signals direct a bit stream decoder 204 to receive the requested bit stream from the bit stream server 250 and to decode a motion video signal from the bit stream. The applet 212 transmits the decoder control signals to the decoder 204 to cause the decoder 204 to receive the bit stream and to decode the motion video signal from the bit stream.

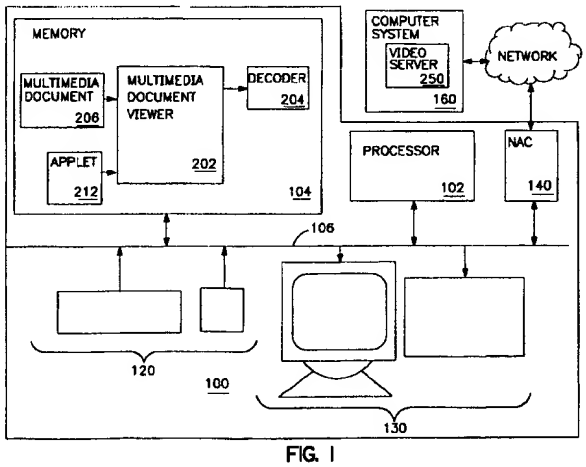

FIG. 1

EP 0 803 826 A2

Description

FIELD OF THE INVENTION

The present invention relates to computer graphical display of motion video and, in particular, to a method and apparatus for facilitating inclusion of motion video in multimedia computer displays.

BACKGROUND OF THE INVENTION

Video servers, including networked video servers, transmit "bit streams" to a video client. Such bit streams, which are sometimes referred to as "streams," generally represent video and/or audio signals which represent titles in a library of multimedia sources. Examples of titles of such a library typically include recordings of motion pictures. In general, a video server receives from a video client a request for a particular title and transmits a stream of the particular title to the video client. An example of a video client is a set top box which is generally known and which decodes the stream received from the video server and transmits the decoded signal to a connected television. The requesting of a particular title, receiving the stream of the particular title, and decoding the stream for display on a television are collectively and generally referred to as video on demand.

Examples of such video on demand servers are described in U.S. Patent Application Serial Number 08/572,639, filed December 14, 1995 by Kallol Mandal and Steven Kleiman and entitled "Method and Apparatus for Delivering Simultaneous Constant Bit Rate Compressed Video Streams at Arbitrary Bit Rates with Constrained Drift and Jitter" (hereinafter the '639 Application) and in U.S. Patent Application Serial Number 08/572,648, filed December 14, 1995 by Kallol Mandal and Steven Kleiman and entitled "Method and Apparatus for Distributing Network Bandwidth on a Video Server for Transmission of Bit Streams Across Multiple Network Interfaces Connected to a Single Internet Protocol (IP) Network" (hereinafter the '648 Application). Both the '639 Application and the '648 Application are incorporated herein in their entirety by reference.

The popularity of the Internet global network is growing extremely rapidly, and perhaps the most popular protocol of the Internet is the Hyper Text Transfer Protocol (HTTP) of the World Wide Web. According to the HTTP protocol of the World Wide Web, documents, which are generally referred to as "pages," incorporate text, graphical images, sound, and motion video which, when viewed, form a multimedia presentation to user. Such pages are typically viewed using a World Wide Web browser, which is a computer process capable of retrieving HTTP pages and presenting the contents of such pages to a user of a computer system through output devices such as a computer video display device and a computer audio circuit coupled to one or more audio speakers. An example of a World Wide Web browser is the Netscape browser available from Netscape Communications Corporation of Mountain View, California.

To display motion video, conventional browsers typically (i) transfer to the computer system in which the browser executes an entire data file which includes data representing a title and (ii) subsequently initiate execution of a player computer process which displays the title to the user on a computer display device. The player computer process is separate from the browser and therefore displays the motion video of the title outside of the page displayed by the browser. In addition, transferring the entire data file prior to displaying the motion video of the title delays substantially the display of the motion video since such data files are typically quite large, e.g., typically 1.8 gigabytes of data to represent a two-hour, VHS-quality motion picture.

Currently, no browser is capable of seamlessly integrating motion video streams into a page of the World Wide Web.

SUMMARY OF THE INVENTION

In accordance with the present invention, a computer process which requests streams of motion video titles and decodes and displays the motion video signals of the stream for display in a computer display device is constructed in the form of an applet of a multimedia document viewer such as a World Wide Web browser. Accordingly, a designer of multimedia documents such as HTML pages can easily incorporate motion video titles into such HTML pages by specifying a few parameters of a desired title or a desired portion of a title to be requested from a video server. The specification of the parameters is in the general form of a well-known parameter specification format dictated by the particular interface of the computer instruction language in which the applet is written.

The applet builds bit stream control signals from the specification of the title or the portion of the title. The bit stream control signals request transmission of the title or the portion of the title from a bit stream server such as a video server

and are in a form appropriate for processing by the bit stream server. The applet transmits the bit stream control signals to the bit stream server to thereby request that the bit stream server initiate transmission of a bit stream representing the requested title or the requested portion of the title.

The applet also builds decoder control signals from the specification of the title or the portion of the title. The decoder control signals direct a bit stream decoder to receive the requested bit stream from the bit stream server and to decode a motion video signal from the bit stream. The applet transmits the decoder control signals to the decoder to cause the decoder to receive the bit stream and to decode the motion video signal from the bit stream.

By using an applet of a multimedia document viewer to request and control receipt by a decoder of a motion video bit stream and to control decoding of the motion video bit stream by the decoder, a designer of a multimedia document can easily and conveniently include motion video images in multimedia documents. In addition, since the applet transmits bit stream control signals to a video server, the motion video signals which can be incorporated into a multimedia document are any such motion video signals stored in such a video server. Such video servers will likely include a large number and wide variety of motion video signals, thereby providing a wealth of motion video content for inclusion in multimedia documents.

The present invention will now be further described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a block diagram of a computer system which is connected to a video server through a network and which includes a multimedia document viewer which in turn processes an applet to include motion video images in a representation of a multimedia document in accordance with the presenting invention.

Figure 2 is a block diagram showing the multimedia document viewer, applet, and video server of Figure 1 in greater detail.

Figure 3 is a block diagram of an applet tag of Figure 2 in greater detail.

Figure 4 is a block diagram of the applet of Figure 2 in greater detail.

DETAILED DESCRIPTION

In accordance with the present invention, a multimedia document 206 (Figure 2) includes an applet 214 which causes a multimedia document viewer 202 to execute an applet 212. Execution of applet 212 requests transmission of a bit stream of a particular title from a video server 250 and controls receipt and decoding of the bit stream by a decoder 204. Decoder 204, in response to control signals received from applet 212, decodes the received bit stream to produce a motion video image and displays the motion video image as an integral part of the representation of multimedia document 206. To include a motion video image as an integral part of a multimedia document, a designer of the multimedia document simply includes in the multimedia document an applet tag, e.g., applet tag 214, which specifies (i) applet 212, (ii) video servoer 250 as the source of a bit stream, and (iii) the particular bit stream to request from video server 250. A brief description of the operating environment of multimedia document viewer 202 and applet 212 facilitates appreciation ofthe present invention.

Figure 1 is a block diagram of a computer system 100 which is generally of the architecture of most computer systems available today. Computer system 100 includes a processor 102 which fetches computer instructions from a memory 104 through a bus 106 and executes those computer instructions. In executing computer instructions fetched from memory 104, processor 102 can retrieve data from or write data to memory 104, display information on one or more computer display devices 130, or receive command signals from one or more user-input devices 120. Processor 102 can be, for example, any of the SPARC processors available from Sun Microsystems, Inc. of Mountain View, California. Memory 104 can include any type of computer memory including, without limitation, randomly accessible memory (RAM), read-only memory (ROM), and storage devices which include magnetic and optical storage media such as magnetic or optical disks. Computer 100 can be, for example, any of the SPARCstation workstation computer systems available from Sun Microsystems, Inc. of Mountain View, California.

Sun, Sun Microsystems, the Sun Logo, Java and Hot Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

Computer display devices 130 can include generally any computer display device such as a printer, a cathode ray tube (CRT), light-emitting diode (LED) display, or a liquid crystal display (LCD). User input devices 120 can include generally any user input device such as a keyboard, a keypad, an electronic mouse, a trackball, a digitizing tablet, thumbwheels, a light-sensitive pen, a touch-sensitive pad, or voice-recognition circuitry.

Computer system 100 also includes network access circuitry 140 which is coupled to processor 102 and memory 104 through bus 106 and which is coupled to a network 150. In accordance with control signals received from processor 102 through bus 106, network access circuitry 140 coordinates transfer of data through network 150 between network access circuitry 140 and similar network access circuitry (not shown) in computer 100B or other computer systems

coupled to computer system 100 through network 150. The transfer of data through network 150 is conventional. Since a video stream representing a VHS-quality motion picture encoded in MPEG-1 format has a bit rate of approximately 1.5 Mbit/second to 2 Mbit/second, a useful minimum threshold is that network access circuitry 140 is capable of receiving data at a rate of at least 2 Mbit/second. Higher quality motion video images have bit rates as high as 8 Mbit/second or higher. Therefore, in one embodiment, network access circuitry 140 is capable of receiving data at a rate of at least 8 Mbit/second. Network access circuitry 140 can be generally any circuitry which is used to transfer data between a computer system and network such as computer system 100 and network 150 and can be, for example, an Ethernet controller chip.

A number of computer processes execute in processor 102 from memory 104, including a multimedia document viewer 202 and a decoder 204. Multimedia document viewer 202 is a computer process which reads a multimedia document 206 and displays the multimedia information specified in multimedia document 206 in one or more of computer display devices 130. In one embodiment, multimedia document 206 is a document in HTML format and multimedia document viewer 202 is an HTML viewer such as the Netscape World Wide Web browser available from Netscape Communications Corporation of Mountain View, California. Multimedia document viewer 202 and multimedia document 206 are shown in greater detail in Figure 2.

Multimedia document viewer 202 retrieves data and tags from a multimedia document such as multimedia document 206. A tag is data which is not itself substantive content of a multimedia document but instead provides format information and can include specification of substantive content which is to be included in the multimedia document and which is located in memory 104 outside of multimedia document 206. For example, a tag can specify a file stored in memory 104 as containing a graphical image which is to be included as substantive content of multimedia document 206. The data and tags of multimedia document 206 collectively define the composition, including substantive content and formatting, of multimedia document 206; and multimedia document viewer 202 displays such substantive content in one or more of computer display devices 130 (Figure 1) in accordance with the data and tags of multimedia document 206. In one embodiment, multimedia document 206 is an HTML document, and the data and tags of multimedia document 206 comport with the HTML language. Multimedia document 206 includes an applet tag 214 (Figure 2) which specifies an applet 212 and a number of operational characteristics of applet 212 as described more completely below.

Multimedia document viewer 202 includes an applet interpreter 210 which retrieves from applet 212 computer instructions and translates such computer instructions into computer instructions of a form appropriate for execution by processor 102 (Figure 1) and submits the translated computer instructions to processor 102 for execution. In one embodiment, applet interpreter 210 (Figure 2) translates and submits for execution a single computer instruction of applet 212 prior to translation and submission for execution of a subsequent computer instruction of applet 212. Applet interpreter 210 can be, for example, the Java applet interpreter or the Hot Java World Wide Web browser available from Sun Microsystems, Inc. and, in such an embodiment, applet 212 comports with the Java computer instruction language interpreted by the Java applet interpreter. As described more completely below, applet 212 is a novel applet which, when executed by processor 102 (Figure 1) through applet interpreter 210 (Figure 2), requests a title from a video server 250 and causes the received bit stream representing the requested title to be decoded in a decoder 204 and displayed in a computer display device as an integral part of a multimedia display of multimedia document 206.

In executing the computer instructions of applet 212, applet interpreter 210 transmits, through network 150 (Figure 1), control signals to an applications programming interface (API) 252 (Figure 2) of a video server 250 which executes within a computer system 160 (Figure 1). Illustrative examples of video server 250 of computer system 160 are described in the '639 and '648 Applications. API 252 (Figure 2) of video server 250 implements a remote procedure calling (RPC) protocol in which API 252 controls video server 250 in response to control signals received by API 252. For example, in response to control signals which request a title and which are transmitted to API 252 by applet interpreter 210, API 252 causes a bit pump 254 of video server 250 to initiate transmission through network 150 (Figure 1) to decoder 204 (Figure 2) of a bit stream representing the requested title. In addition, API 252 can transmit to applet interpreter 210 status information regarding a title stored within video server 250 or regarding a bit stream transmitted by bit pump 254 in response to control signals requesting such status information.

Decoder 204 is a computer process executing within processor 102 (Figure 1) from memory 104. Decoder 204 receives data representing a motion video display encoded in a particular format. In one embodiment, decoder 204 is the MPEG Expert (MPX) decoder available from Applied Vision and decodes motion video signals according to the MPEG-1 encoding format. Applet interpreter 210 transmits to decoder 204 control signals which control the decoding by decoder 204 of the bit stream received from bit pump 254 of video server 250. Specifically, applet interpreter 210 transmits to decoder 204 control signals directing decoder 204 to start or stop decoding the bit stream received from bit pump 254 or specifying characteristics of the bit stream received from bit pump 254 such as the bit rate, encoding format, and the coordinates of a particular location within one or more of computer display devices 130 (Figure 1) in which to display the decoded motion video images. In addition, applet 212 determines which communications port through network access circuitry 140 (Figure 1) the bit stream is to be received and transmits to decoder 204 (Figure 2) control signals identifying the selected communications port. Applet 212 can therefore determine which communi-

cations ports are used by other applications and can avoid conflicts resulting from access of decoder 204 of a communications port by selecting a communications port which is not used by another computer process of computer system 100 (Figure 1).

Applet tag 214 is shown in greater detail in Figure 3. Applet tag 214 includes a number of fields which collectively define a bit stream to be received and decoded for display by decoder 204 (Figure 2). A field is a collection of data which collectively define a item of information. Applet tag 214 includes (i) an applet identifier field 302, (ii) a width field 304, (iii) a height field 306, (iv) a server identifier field 308, and (v) an encoding format field 310. Applet tag 214 can also include any of the following optional fields: (vi) a title field 312, (vii) an image field 314, (viii) a play/pause field 316, (ix) a start field 318, and (x) a duration field 320.

Applet identifier field 302 specifies applet 212 as the applet to be retrieved and executed by applet interpreter 210. Width field 304 and height field 306 specify the width and height, respectively, in display coordinate space of a computer display device, i.e., specify the size of the viewport in which the decoded motion video image is displayed. Server identifier field 308 specifies video server 250 (Figure 2) as the source of the desired bit stream. Encoding format field 310 (Figure 3) specifies the particular encoding format, e.g., MPEG1SYS encoding format, ofthe bit stream received by decoder 204 (Figure 2). Title field 312 (Figure 3) specifies the particular title to be retrieved from server 250 (Figure 2). Alternatively, title field 312 can specify the address of a multicast bit stream.

Image field 314 (Figure 3), if included, specifies a still video image to be displayed in the space specified by width field 304 and height field 306 if the title specified by title field 312 is unavailable. Play/pause field 316, if included, specifies whether the motion video image received from video server 250 (Figure 2) is initially in a play state or in a paused state. Start field 318 (Figure 3), if included, specifies an offset into the title of a portion of the title, i.e., the point within the title at which the bit stream should begin. For example, start field 318 can specify that the requested bit stream begin at 3 minutes and 10 seconds into the title. Duration field 320, if included specifies the duration of a desired portion of the title. For example, duration field 320 can specify that a 30-minute portion of the title is requested. In one embodiment, start field 318 and duration field 320 are specified in terms of an integer number of nanoseconds.

Thus, by specifying the few fields described above and shown in Figure 3, a designer of multimedia document 206 can include as an integral part of multimedia document 206 a motion video image retrieved from video server 250. The following is an illustrative example of applet tag 214 in HTML format.

```
<applet code="SunMediaCenterPlayer.class" width=704 height=520>
<param name=port value="1973">
<param name=format value="MPEG1SYS">
<param name=host value="sqas-6">
<param name=img value="/images/bkgx.gif">
</applet>
```

Applet 212 (Figure 2) includes computer instructions which, when executed, request a title from video server 250 and control decoding and display of the decoded motion video signals by decoder 204 and is shown in greater detail in Figure 4. The computer instructions of applet 212 are organized into various levels, each of which defines a respective component of the behavior of applet 212. Applet 212 includes a player level 402, an API level 404, a decoder level 406, and a detailed decoder level 408.

Player level 402 includes computer instructions which, when executed, implement a graphical user interface in which a user can control the bit stream received by video server 250 (Figure 2) and the display of the decoded motion video signals of the bit stream by physical manipulation of one or more of user input devices 120 (Figure 1). In one embodiment, the computer instructions of player level 402 (Figure 4), when executed, cause graphical and/or textual representation of control mechanisms to be displayed in one or more of computer display devices 130 (Figure 1). Such control mechanisms are known and conventional and include, without limitation, virtual buttons, pull-down menus, virtual radio buttons, virtual check boxes, and sliding scroll bars. In a conventional manner, a user activates one or more of such control mechanisms by physical manipulation of one or more of user input devices 120 (Figure 1) and such physical manipulation results in receipt by player level 402 (Figure 4) of applet 212 of signals and/or data representing such activation.

API level 404 includes computer instructions which, when executed, implement the RPC protocol of API 252 (Figure 2) of video server 250 and invoke RPC calls to API 252 to control the bit stream transmitted by bit pump 254 in accordance with interaction of a user with the graphical user interface implemented by player level 402 (Figure 4).

Decoder level 406 and detailed decoder level 408 collectively control operation of decoder 204 (Figure 2), generally controlling the decoding of the bit stream received from video server 250 by decoder 204 and the display in a computer display device of the decoded motion video image. Decoder level 406 includes computer instructions and data structures which are not specific to any particular decoder, while detailed decoder level 408 includes computer instructions and data structures which are specific to decoder 204. It is generally preferred that detailed decoder level 408 is as

small and simple as possible such that the majority of computer instructions of decoder levels 406 and 408 are included in decoder level 406. Accordingly, adapting applet 212 (Figure 2) to operate in conjunction with a decoder other than decoder 204 requires modification of only detailed decoder level 408 and, therefore, as little modification as possible.

Appendix A is a computer source code listing of a preferred embodiment of applet 212. The modules of Appendix A are written in the Java applet computer instruction language developed by Sun Microsystems, Inc. of Mountain View, California. The computer instructions of the Java applet computer instruction language are object-oriented, and each of the modules of Appendix A represents a respective class of objects. Player level 402 (Figure 4), in this embodiment, includes classes SunMediaCenterPlayer, Player, and PositionSlider as defined in the computer source code listing of Appendix A. API level 404, in this embodiment, includes classes MsmPlayer, MsmSession, MsmAccessRight, MsmPersistence, MsmPlaylist, MsmToString, MsmItem, MsmTitleItem, MsmDeadAirItem, MsmException, XdrBlock, and PortMapper as defined in the computer source code listing of Appendix A. Decoder level 406, in this embodiment, includes classes Decoder and DecoderImpl as defined in the computer source code listing of Appendix A. Detailed decoder level 408, in this embodiment, includes class MpxDecoderImpl as defined in the computer source code listing of Appendix A.

In the preferred embodiment of the present invention defined by Appendix A, a module "loop" includes computer instructions of the C computer instruction language and defines a loop computer process which executes independently of multimedia document viewer 202 (Figure 2). The loop computer process cooperates with multimedia document viewer 202 and decoder 204 to request and receive from video server 250 bit streams representing multicast motion video signals.

The above description is illustrative only and is not limiting. The present invention is therefore defined solely and completely by the appended claims together with their full scope of equivalents.

APPENDIX A

SunMediaCenterPlayer

```
/*
 * @(#)SunMediaCenterPlayer.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

import java.applet.*;
import java.awt.*;
import java.net.*;
import java.io.*;
import COM.Sun.isg.smcjc.*;

public class SunMediaCenterPlayer extends Applet {
    private Player player;
    private TextArea reporter;
    private Thread thread;

    public SunMediaCenterPlayer() {
     setLayout(new BorderLayout());
     player = new Player();
     add("Center", player);
    }

    public synchronized void init() {
     if (reporter != null && reporter.getParent() == this) {
         remove(reporter);
         reporter.setText("");
         validate();
     }
     try {
         int port=getParameterInt("port",-1);
         int vc=getParameterInt("vc",-1);
            if (vc!=-1){
             player.init(
             getParameterRequired("host"),
             getParameterRequired("title"),
```

```
                getParameterLong("start", 0L),
                getParameterLong("duration", 0L),
                getParameterString("loop",
"false").equalsIgnoreCase("true"),
                getParameterString("cmd", "play"),
                getParameterImage("img", null),
                      vc,"",
                      getParameterURL("CC"),
                getParameterRequired("interface"));
               }else{
                if (port==-1){
                player.init(
                getParameterRequired("host"),
                getParameterRequired("title"),
                getParameterLong("start", 0L),
                getParameterLong("duration", 0L),
                getParameterString("loop",
"false").equalsIgnoreCase("true"),
                getParameterString("cmd", "play"),
                getParameterImage("img", null),
                      port,"",
                      getParameterURL("CC"),null);
                 }else{
                player.init(
                getParameterRequired("host"),
                "none",0L,0L,false,"play",
                getParameterImage("img", null),
                      port,
                getParameterRequired("format"),
                      getParameterURL("CC"),null);
                 }
               }
        } catch (IOException e) {
            report(e, "parsing Sun MediaCenter player parameters");
        }
      }

      public synchronized void start() {
       try player.start(); catch (IOException e)
           report(e, "starting a Sun MediaCenter player");
      }

      public synchronized void stop() {
       try player.stop(); catch (IOException e)
           report(e, "stopping a Sun MediaCenter player");
      }
```

```
    private String getParameterRequired(String key) throws
IOException {
     String val = getParameter(key);
     if (val != null) return val;
     throw new IOException("missing required parameter " + key);
    }

    private int getParameterIntRequired(String key) throws
IOException {
     String val = getParameter(key);
     if (val != null)
           try return Integer.parseInt(val); catch
(NumberFormatException e)
              throw new IOException(
                 "parameter " + key + " is not a valid int: " +
val);
;
     throw new IOException("missing required parameter " + key);
    }

    private URL getParameterURL(String key) {
        URL res=null;
     String val = getParameter(key);
     if (val == null) return null;
        try res=new URL(val);
          catch (MalformedURLException e) try res=new
URL(getDocumentBase(),val);
           catch (MalformedURLException f)
System.out.println("MalformedURLException");
        return res;
    }

    private String getParameterString(String key, String dflt) {
     String val = getParameter(key);
     if (val == null) return dflt;
     return val;
    }

    private int getParameterInt(String key, int dflt) throws
IOException {
     String val = getParameter(key);
     if (val == null) return dflt;
     try return Integer.parseInt(val); catch
(NumberFormatException e)
         throw new IOException(
          "parameter " + key + " is not a valid int: " + val);
    }
```

```
    private long getParameterLong(String key, long dflt) throws
IOException {
     String val = getParameter(key);
     if (val == null) return dflt;
     try return Long.parseLong(val); catch (NumberFormatException
e)
         throw new IOException(
          "parameter " + key + " is not a valid long: " + val);
    }

    private Image getParameterImage(String key, Image dflt) {
     String val = getParameter(key);
     if (val == null) return dflt;
     return getImage(getDocumentBase(), val);
    }

    private synchronized void report(Exception e, String doing) {
     ByteArrayOutputStream os = new ByteArrayOutputStream();
     PrintStream ps = new PrintStream(os);
     ps.print("An error occurred while ");
     ps.print(doing);
     ps.println(":");
     e.printStackTrace(ps);
     if (reporter == null) {
         reporter = new TextArea("");
         reporter.setEditable(false);
     }
     reporter.appendText(os.toString());
     if (reporter.getParent() != this) {
         add("North", reporter);
         validate();
     }
    }

}
```

Player

```
/*
 * @(#)Player.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.1sc
 * author Christopher Lindblad    ( Msm API & Mpx API )
 * author Stephane CACHAT         (Closed Caption & Multicasting)
 *
 */

package COM.Sun.isg.smcjc;

import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;

public class Player extends Panel implements Runnable {
    private long playDuration;
    private long startOffset;
    private long seekPosition;
    private long tellPosition;
    private double tellPositiond;
    private MsmPlayer player;
    private String host;
    private String titleName;
    private String msg;
    private String format;
    private Image img;
    private Thread thread;
    private Panel controlLine;
    private Panel controlButtons;
    private TextArea reporter;
    private Decoder decoder;
    private PositionSlider positionSlider;
    private Button[] buttons;
    private int cmd = 999;
    private int initialCmd;
    private int port;
    private boolean loop;
    private boolean Msm;
    private URL CC;
    private List CCt;
```

```
        private int CCz=0;
        private String[] CCb=new String[1024];
        private Double[] CCi=new Double[1024];
        private int CCl=0;
        private int CCo=0;
        private int CCm=0;
        private boolean playing = false;
        private TextField CCs;
        private String ATM;

        public Player() {
         setLayout(new BorderLayout());
         decoder = new Decoder();
         add("Center", decoder);
        }

       public synchronized void init(
         String host, String titleName,
         long startOffset, long playDuration, boolean loop,
         String cmd, Image img,int port,String format,URL CC,String
ATM)
         throws IOException {
                URLConnection uc;
                Double d;
                String str;
                int i=0;
                int j=0;

                this.port=port;
                if ((port!=-1)&&(ATM==null)){
                    Msm=false;
                }else{
                    Msm=true;
                 this.initialCmd = parseCmd(cmd);
                }
                this.CC=CC;
                this.ATM=ATM;
            this.host = host;
            this.titleName = titleName;
            this.startOffset = startOffset;
            this.playDuration = playDuration;
            this.loop = loop;
            this.img = img;
            this.format = format;
               if (CC!=null){
                 CCt= new List();
                 CCt.minimumSize(6);
```

```
                CCt.preferredSize(6);
                uc= CC.openConnection();
                DataInputStream in=new
DataInputStream(uc.getInputStream());
                str="-";
             CCb[i]=new String("*");
                CCi[i]=new Double(0.0);
                i++;
                while (in.available()>0){
                  str=in.readLine();
                  while
((str.trim().length()==0)&&(in.available()>0)) str=in.readLine();
                  if (str!=null){
                    j=str.trim().indexOf(' ');
                    if (j>0){
                CCb[i]=new String(str.substring(j+1)).trim();
                      CCt.addItem(CCb[i]);
                      if (CCb[i]==null) CCb[i]="*";
                     CCi[i]=new Double(str.substring(0,j).trim());
                      i++;
                 }
                  }
                }
                CCm=i-1;
                in.close();
              }
      }

    public synchronized void start() throws IOException {
      if (reporter != null && reporter.getParent() == this) {
          remove(reporter);
          reporter.setText("");
          validate();
      }
      if (thread == null) {
          cmd = initialCmd;
          thread = new Thread(this);
          thread.start();
      }
    }

    public synchronized void stop() throws IOException {
      if (thread != null) {
          thread = null;
          notify();
      }
    }
```

```
    public synchronized boolean action(Event evt, Object arg) {
     if (buttons != null && evt.target instanceof Button) {
         Button b = (Button)evt.target;
         for (int i = 0; i < buttons.length; i++) {
          if (b == buttons[i]) cmd = i;
         }
         notify();
     };
     if (CC != null && evt.target ==CCt) {
         seekPosition = (long)(new
Double(CCi[CCt.getSelectedIndex()].doubleValue()*10).intValue())*
100000000;
         cmd = SEEK;
         notify();
        };
     if (CC != null && evt.target==CCs) {
            if (CCl<CCm){
              CCz=CCl+1;
            }else{
              CCz=0;
            };

while((CCz!=CCl)&&(CCb[CCz].indexOf(CCs.getText())<0)) {
              CCz++;
              if (CCz>CCm) CCz=0;
            }
            if (CCb[CCz].indexOf(CCs.getText())>=0){
              CCt.select(CCz);
              CCt.makeVisible(CCz+1);
           seekPosition = (long)(new
Double(CCi[CCt.getSelectedIndex()].doubleValue()*10).intValue())*
100000000;
           cmd = SEEK;
           notify();
            }
       }
     return true;
    }

    private void setConnect(MsmConnect connect) throws
IOException {
     try {
         player.setConnect(connect);
     } catch (MsmException e) {
         /* Try it with destTiAddr in beta 0.5 syntax. */
System.out.println("DesTiAddr="+connect.destTiAddr);
         InputStream is = new
```

```
StringBufferInputStream(connect.destTiAddr);
          StreamTokenizer st = new StreamTokenizer(is);
          String host;
          int udpport;
             if(ATM==null){
            if (st.nextToken() == StreamTokenizer.TT_WORD &&
           st.sval.equals("host") &&
           st.nextToken() == '=' &&
           st.nextToken() == StreamTokenizer.TT_WORD &&
           (host = st.sval) != null &&
           st.nextToken() == ',' &&
           st.nextToken() == StreamTokenizer.TT_WORD &&
           st.sval.equals("udpport") &&
           st.nextToken() == '=' &&
           st.nextToken() == StreamTokenizer.TT_NUMBER &&
           (udpport = (int)st.nval) != 0) {
           connect.destTiAddr = "be0,"+host+","+udpport;
           player.setConnect(connect);
            } else {
           throw e;
            }
            }else{
            throw e;
            }
      }
     }

     public synchronized void run() {
      Thread currentThread = Thread.currentThread();
      MsmSession session = null;
      MsmTitle title = null;
         MsmItem[] items = null;
         int speed=0;


      if (Msm){
             controlButtons = new Panel();
          controlButtons.setLayout(new FlowLayout());
          controlButtons.add(cmds[PAUSE], new
Button(labels[PAUSE]));
          controlLine = new Panel();
          controlLine.setLayout(new BorderLayout());
          controlLine.add("East", controlButtons);
          positionSlider = new PositionSlider(this);
          controlLine.add("Center", positionSlider);
          add("South", controlLine);
             if (CC!=null){
```

```
                    Panel CCp=new Panel();
                    CCp.setLayout(new BorderLayout());
                    Panel CCq=new Panel();
                    CCq.setLayout(new BorderLayout());

                    CCs= new TextField(15);
                    CCs.isEditable();
                CCq.add("South", CCs);
                    Label l=new Label("Search");
                CCq.add("Center", l);
                    CCp.add("East",CCq);
                    CCp.add("Center",CCt);
                controlLine.add("North",CCp);
                }
            }
        try {
            if (Msm){
                    items = new MsmItem[1];
                session = new MsmSession(host);
                title = session.getTitleStatus(titleName);
                if (playDuration == 0L) playDuration =
title.totalPlayDuration;
                    format=title.format;
                }
            decoder.init(format, img,host,port,ATM);
            if (Msm){
                    titleInit(title);
                player = new MsmPlayer(session, info(),
MsmPlayer.TIME_MAXTIME);
                player.setPersistence(new MsmPersistence(
                 MsmPersistence.TYPE_NONE,
                 MsmPlayer.TIME_MAXTIME));
                items[0] = new MsmTitleItem(
                 titleName, playDuration, startOffset, playDuration,
                 playDuration, false, true, title.maxBitRate);
                player.setPlaylist(new MsmPlaylist(
                 MsmPlayer.TIME_CURRENT, loop, 0,
MsmPlayer.TIME_MAXTIME,
                 items, 0, 0));
                setConnect(new MsmConnect(
                 decoder.destTiAddr(), decoder.encap(),
title.maxBitRate));
                playing = false;
                speed = MsmPlayer.SPEED_FORWARD;
               }else{
                invalidate();
                validate();
```

```
            }
          while (currentThread == thread) {
           switch (cmd) {
           case NOP: {
               if (Msm) {
                        MsmPlayStatus status =
player.getPlayStatus();
                    if (tellPosition != status.currentPosition) {
                     tellPosition = status.currentPosition;
                     positionSlider.repaint();
                    }

tellPositiond=(tellPosition/1000000000)+3.0;
                          if (CC!=null){
                            CCo=CCl;
                            while
((CCi[CCl+1].doubleValue()<tellPositiond)&&(CCl+1<CCm))  CCl++;
                            while
((CCi[CCl].doubleValue()>tellPositiond)&&(CCl>0)) CCl--;
                            if (CCo!=CCl) {
                               CCt.select(CCl-1);
                               CCt.makeVisible(CCl);
                            }
                          }
                    player.setPersistence(new MsmPersistence(
                     MsmPersistence.TYPE_NONE,
                     status.currentDate+60*1000000000L));
               }
                     break;
           }
           case PAUSE: {
               decoder.pause();
               if (Msm) player.pause(MsmPlayer.TIME_CURRENT);
               decoder.flush();
               playing = false;
               decoder.play();
               break;
           }
           case GOTO_START: {
               tellPosition = 0L;
               if (Msm) positionSlider.repaint();
               decoder.stop();
               if (Msm) player.play(MsmPlayer.SPEED_FORWARD,
                     0L,
                     0L,
                     MsmPlayer.TIME_CURRENT);
               decoder.flush();
```

```
            break;
        }
        case GOTO_END: {
            tellPosition = playDuration;
            if (Msm) positionSlider.repaint();
            decoder.stop();
            if (Msm) player.play(MsmPlayer.SPEED_REVERSE,
                  playDuration,
                  0L,
                  MsmPlayer.TIME_CURRENT);
            decoder.flush();
            break;
        }
        case SEEK: {
            tellPosition = seekPosition;
            if (Msm) positionSlider.repaint();
            if (playing) {
             decoder.flush();
             if (Msm) player.play(speed,
                      seekPosition,
                      MsmPlayer.TIME_MAXTIME,
                      MsmPlayer.TIME_CURRENT);
            } else {
             long duration = SEEKDURATION;
             long position = seekPosition-duration;
             if (position < 0L) {
                 duration += position;
                 position -= position;
             }
             decoder.play();
             decoder.flush();
             if (Msm) player.play(MsmPlayer.SPEED_FORWARD,
                      position,
                      duration,
                      MsmPlayer.TIME_CURRENT);
            }
            break;
        }
        default: {
            decoder.play();
            decoder.flush();
            if (Msm){
                      speed = cmd;
                player.play(speed,
                  MsmPlayer.TIME_CURRENT,
                  MsmPlayer.TIME_MAXTIME,
                  MsmPlayer.TIME_CURRENT);
```

```
                    playing = true;
                          if (CC!=null)
                            if (CCo!=CCl) {
                               CCt.select(CCl-1);
                               CCt.makeVisible(CCl);
                            }
                        }
                }
                }
                cmd = NOP;
                try wait(100); catch (InterruptedException e);
              }
        } catch (Exception e) {
            report(e, "communicating with a Sun MediaCenter
server");
        } finally {
            try {
             try decoder.stop(); catch (Exception e)
                 report(e, "stopping a video decoder");
                   if (Msm){
                 if (player != null) {
                     try player.delete(); catch (Exception e)
                      report(e, "deleting a Sun MediaCenter
player");
                     player = null;
                       }
                }
            } finally {
                   if(Msm){
                 if (session != null) {
                     try session.close(); catch (Exception e)
                      report(e, "closing a Sun MediaCenter
connection");
                 }
                   }
            }
        }
      }

      /*
       * Callback from the PositionSlider.
       * Unsynchronized to avoid deadlock.
       * @return value between 0 and 1 indicating where in the file
we are.
       */
      public double tell() {
       if (playDuration == 0L) return 0.0D;
```

```
        return (double)tellPosition / (double)playDuration;
      }

      /*
       * Callback from the PositionSlider.
       * Seek to a relative position in a file.
       * @param position Value between 0 and 1
       * indicating where in the file to go.
       */
      public synchronized void seek(double position) {
       if (playDuration == 0) return;
       seekPosition = (long)(position*playDuration);
       cmd = SEEK;
       notify();
      }

      private String info() throws UnknownHostException {
          String hostName =
InetAddress.getLocalHost().getHostName();
          String javaVersion = System.getProperty("java.version");
          String javaVendor = System.getProperty("java.vendor");
          String osArch = System.getProperty("os.arch");
          String osName = System.getProperty("os.name");
          String osVersion = System.getProperty("os.version");
          return hostName
              + " Java " + javaVersion + " (" + javaVendor + ")"
              + " (" + osArch + " " + osName + " " + osVersion +
")";
      }

      private void addButton(int i) {
       buttons[i] = new Button(labels[i]);
       controlButtons.add(cmds[i], buttons[i]);
      }

      /**
       * Initialize for a title.
       * @param title The title to play.
       */
      private void titleInit(MsmTitle title) throws IOException {
       controlButtons.removeAll();
       buttons = new Button[labels.length];
       for (int i = MsmPlayer.SPEED_SLOWEST_FORWARD;
            i <= MsmPlayer.SPEED_SCENE_FORWARD;
            i++) {
           if (title.speedScale[i] != 0) {
            addButton(GOTO_START);
```

```
            break;
          }
        }
        for (int i = MsmPlayer.SPEED_SCENE_REVERSE;
             i <= MsmPlayer.SPEED_SLOWEST_REVERSE;
             i++) {
          if (title.speedScale[i] != 0) addButton(i);
        }
        addButton(PAUSE);
        for (int i = MsmPlayer.SPEED_SLOWEST_FORWARD;
             i <= MsmPlayer.SPEED_SCENE_FORWARD;
             i++) {
          if (title.speedScale[i] != 0) addButton(i);
        }
        for (int i = MsmPlayer.SPEED_SCENE_REVERSE;
             i <= MsmPlayer.SPEED_SLOWEST_REVERSE;
             i++) {
          if (title.speedScale[i] != 0) {
            addButton(GOTO_END);
            break;
          }
        }
        /* recompute layout */
        controlLine.invalidate();
        invalidate();
        validate();
        /* resize if we need to */
        Component c = getParent();
        while (c != null) {
          if (c instanceof Applet) {
            Dimension ps = c.preferredSize();
            Rectangle b = c.bounds();
            if (ps.width != b.width || ps.height != b.height) {
              // This wedges Netscape Navigator 2.0
              // c.resize(ps.width, ps.height);
            }
            break;
          }
        }
      }

      private void report(Exception e, String doing) {
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        PrintStream ps = new PrintStream(os);
        ps.print("An error occurred while ");
        ps.print(doing);
        ps.println(":");
```

```
        e.printStackTrace(ps);
        if (reporter == null) {
            reporter = new TextArea("");
            reporter.setEditable(false);
        }
        reporter.appendText(os.toString());
        if (reporter.getParent() != this) {
            add("North", reporter);
            validate();
        }
    }

    private int parseCmd(String cmd) throws IOException {
        for (int i = 0; i < cmds.length; i++) {
            if (cmd.equalsIgnoreCase(cmds[i])) return i;
        }
        throw new IOException("Not a valid Player command: "+cmd);
    }

    private static final long SEEKDURATION = 4000000000L;

    private static final int PAUSE = 16;
    private static final int GOTO_START = 17;
    private static final int GOTO_END = 18;
    private static final int SEEK = 19;
    private static final int NOP = 20;

    private static final String[] labels = {
        "|<<<<",        // MsmPlayer.SPEED_SCENE_REVERSE
        "<<<<",              // MsmPlayer.SPEED_FASTEST_REVERSE
        "<<<",               // MsmPlayer.SPEED_FASTER_REVERSE
        "<<",                // MsmPlayer.SPEED_FAST_REVERSE
        "<",            // MsmPlayer.SPEED_REVERSE
        "|<",                // MsmPlayer.SPEED_SLOW_REVERSE
        "||<",               // MsmPlayer.SPEED_SLOWER_REVERSE
        "|||<",              // MsmPlayer.SPEED_SLOWEST_REVERSE
        ">|||",              // MsmPlayer.SPEED_SLOWEST_FORWARD
        ">||",               // MsmPlayer.SPEED_SLOWER_FORWARD
        ">|",                // MsmPlayer.SPEED_SLOW_FORWARD
        ">",            // MsmPlayer.SPEED_FORWARD
        ">>",                // MsmPlayer.SPEED_FAST_FORWARD
        ">>>",               // MsmPlayer.SPEED_FASTER_FORWARD
        ">>>>",              // MsmPlayer.SPEED_FASTEST_FORWARD
        ">>>>|",        // MsmPlayer.SPEED_SCENE_FORWARD
        "||",                // PAUSE
        "||<<<<",       // GOTO_START
        ">>>>||",       // GOTO_END
```

```
        "",             // SEEK
        "",             // NOP
    };

    private static final String[] cmds = {
        "scene_reverse",    // MsmPlayer.SPEED_SCENE_REVERSE
        "fastest_reverse",  // MsmPlayer.SPEED_FASTEST_REVERSE
        "faster_reverse",   // MsmPlayer.SPEED_FASTER_REVERSE
        "fast_reverse",         // MsmPlayer.SPEED_FAST_REVERSE
        "reverse",          // MsmPlayer.SPEED_REVERSE
        "slow_reverse",         // MsmPlayer.SPEED_SLOW_REVERSE
        "slower_reverse",   // MsmPlayer.SPEED_SLOWER_REVERSE
        "slowest_reverse",  // MsmPlayer.SPEED_SLOWEST_REVERSE
        "slowest_forward",  // MsmPlayer.SPEED_SLOWEST_FORWARD
        "slower_forward",   // MsmPlayer.SPEED_SLOWER_FORWARD
        "slow_forward",         // MsmPlayer.SPEED_SLOW_FORWARD
        "play",             // MsmPlayer.SPEED_FORWARD
        "fast_forward",         // MsmPlayer.SPEED_FAST_FORWARD
        "faster_forward",   // MsmPlayer.SPEED_FASTER_FORWARD
        "fastest_forward",      // MsmPlayer.SPEED_FASTEST_FORWARD
        "scene_forward",    // MsmPlayer.SPEED_SCENE_FORWARD
        "pause",        // PAUSE
        "goto_start",       // GOTO_START
        "goto_end",         // GOTO_END
        "seek",             // SEEK
        "nop",          // NOP
    };

}
```

PositionSlider

```
/*
 * @(#)PositionSlider.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.awt.*;
import java.io.*;

class PositionSlider extends Canvas {
    private Player player;
    private int hgap;
    private int vgap;
    private int wid;

    public PositionSlider(Player player) {
     this(player, 5, 5, 6);
    }

    public PositionSlider(Player player, int hgap, int vgap, int
wid) {
     this.player = player;
     this.hgap = hgap;
     this.vgap = vgap;
     this.wid = wid;
    }

    public void update(Graphics g) {
     paint(g);
    }

    public synchronized void paint(Graphics g) {
     Rectangle r = bounds();
     int position = (int)((r.width-hgap*2)*player.tell())+hgap;
     g.setColor(getBackground());
     g.fillRect(0, 0, r.width, vgap*2);
     g.fillRect(0, r.height-vgap*2, r.width, vgap*2);
     g.fillRect(0, vgap*2, r.width-hgap*2, r.height-vgap*2);
```

```
        g.fillRect(r.width-hgap, vgap*2, r.width, r.height-vgap*2);
        g.fill3DRect(hgap, vgap*2, r.width-hgap*2, r.height-vgap*4,
    false);
        g.fill3DRect(position-2, vgap, wid, r.height-vgap*2, true);
      }

      private synchronized void seek(int x) {
        Rectangle r = bounds();
        double position = ((double)(x-hgap)) /
    ((double)(r.width-hgap*2));
        if (position < 0.0D) position = 0.0D;
        if (position > 1.0D) position = 1.0D;
        player.seek(position);
      }

      public boolean mouseDown(Event e, int x, int y) {
        seek(x);
        return true;
      }

      public boolean mouseDrag(Event e, int x, int y) {
        seek(x);
        return true;
      }

    }
```

MsmPlayer

```
/*
 * @(#)MsmPlayer.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.io.*;

/**
 * Media Stream Manager Client API
 *
 * MSM allows for the creation of "players".  A player is a
persistent entity
 * that provides for the scheduled delivery of isochronous data
to a
 * particular destination.  To accomplish this task, a player
maintains a
 * playlist of titles, the state of a "playhead" which traverses
this
 * playlist, and an access list controlling who can perform
various functions
 * on the player.
 *
 * MSM, when supplied with titles that have been prepared for
presentation at
 * multiple presentation rates, manages the position index
lookups and stream
 * switching necessary for "trick play".
 *
 * Associated with a player is a "playhead" that maintains a
destination for
 * the isochronous data (possibly different than the controlling
client) and a
 * "playPosition" which travels along the playlist at the
selected
 * presentation rate and delivers isochronous data as scheduled
to the
 * destination.  The position, presentation rate, and
```

```
presentation direction
 * of the playhead can be controlled via play(), pause(), and
resume(). The
 * initiation of play can be synchronized with "wall clock time"
via play();
 * presentation will then stay synchronized with wall-clock time
as long as
 * presentation rate and direction are Normal-Rate,
Forward-Direction.
 *
 * Latency from invocation of the play() request until actual
start of stream
 * may be reduced by "pre-rolling" with a play() request that has
zero
 * duration. This may also be used to set a current playlist
position without
 * actually starting play.
 *
 * MSM manages concurrent updates to a playlist by returning a
modification
 * timestamp with playlist status. The modification timestamp
indicates the
 * time of the last modification of the playlist. When a client
wishes to
 * update a playlist, the client will first obtain status
containing a
 * modification timestamp to understand the current state of the
playlist.
 * Based on this status, the client then determines the
appropriate updates
 * and passes those updates along with the modification timestamp
of the
 * status on which the updates were based to msm. If msm finds
that the
 * modification timestamp has not changed, implying that the
clients updates
 * are based on currently valid playlist state, the playlist
update will
 * succeed. If the modification timestamp indicates that the
playlist has
 * been modified since this client obtained status, the update
will be
 * rejected. In this case, the client should reobtain status,
reaccess the
 * update, and then if appropriate resubmit the update with the
modification
 * timestamp of the new status. There is a designated timestamp
```

```
that forces
 * playlist modifications, this may be used if some external
method of
 * concurrency control is preferred.
 *
 * MsmPlaylist may be edit while play is in progress.  Normally,
changes to the
 * playlist will not take effect until the current item in play
completes.  A
 * playlist modification can be forced to take effect immediately
by calling
 * resume().  resume() should be called with the speed argument
being the
 * current (or desired new speed) and the startPosition argument
being
 * TIME_CURRENT.  If the contents of the playlist at the current
position of
 * the playhead have not been modified, this call will not
disturb the
 * outgoing data stream.
 *
 * MSM optionally maintains players persistently across server
outages.  When
 * this option is selected, a successful return from a player
request
 * indicates that the player modifications have been made
persistently.
 * Persistent players may optionally restart play on state
recovery, play may
 * be restarted at the last played position or at the position
that the
 * position that play would be add had no outage occurred.
 *
 * Access to read and modify players is controlled by access
control lists
 * associated with the players.  These may be modified by
 * msmPlayerSetAccess().
 *
 * Access rights are "Read", "Control", and "Admin".  Read rights
all state to
 * be seen.  Control rights allow "trick-play" operations to be
controlled.
 * Admin rights allow creation of players, and connection,
access, and
 * persistence attributes of players to be set.  Access rights
are associated
 * with "agents" (eg users) appropriate for the authorization
```

```
mechanism
 * selected.  The reserved agent name "*" represents ALL agents,
those
 * granting a right to "*", grants the right to all agents.
 *
 */
public class MsmPlayer {
    private MsmSession session;
    private byte[] handle;

    /**
     * Creates a player.  The player is initialized
non-persistent.
     * @param session A server session.
     * @param info Saved, but uninterpreted by server. May be
null.
     *   Used to describe the player for administrative purposes.
     * @param terminateDate Date at which player should be
auto-deleted.
     *   If TIME_MAXTIME, the player will never be auto-deleted,
it must
     *   be deleted via delete.
     * @exception IOException If an error has occurred.
     */
    public MsmPlayer(MsmSession session, String info, long
terminateDate)
     throws IOException {
         this.session = session;
         XdrBlock call = session.newCall(PLAYER_CREATE);
         call.xdroutString(info);
         call.xdroutMsmTime(terminateDate);
         XdrBlock reply = session.rpc(call);
         handle = reply.xdrinBytes(HANDLELEN);
         reply.done();
     }

    MsmPlayer(MsmSession session, XdrBlock xdr) {
     this.session = session;
     handle = xdr.xdrinBytes(HANDLELEN);
    }

    void xdrout(XdrBlock xdr) {
     xdr.xdroutBytes(handle,HANDLELEN);
    }

    public MsmSession getSession() {
     return session;
```

```
    }

    public byte[] getHandle() {
     return handle;
    }

    /**
     * Opens an existing player.
     * @param session A server session.
     * @param handle An opaque handle to the player.
     */
    public MsmPlayer(MsmSession session, byte[] handle) {
     this.session = session;
     this.handle = handle;
    }

    /**
     * Deletes the player.  In progress play of the player is
stopped.
     * @exception IOException If an error has occurred.
     */
    public void delete() throws IOException {
     XdrBlock call = session.newCall(PLAYER_DELETE);
     this.xdrout(call);
     session.rpc(call).done();
    }

    /**
     * Modifies access control list for player.
     * @param rights The access modifications.
     * @exception IOException If an error has occurred.
     */
    public void setAccess(MsmAccessRight[] rights) throws
IOException {
     XdrBlock call = session.newCall(PLAYER_SETACCESS);
     this.xdrout(call);
     call.xdroutInt(rights.length);
     for (int i = 0; i < rights.length; i++)
rights[i].xdrout(call);
     session.rpc(call).done();
    }

    /**
     * Get access control list for player.
     * @return The access modifications.
     * @exception IOException If an error has occurred.
     */
```

```
    public MsmAccessRight[] getAccess() throws IOException {
     XdrBlock call = session.newCall(PLAYER_GETACCESS);
     this.xdrout(call);
     XdrBlock reply = session.rpc(call);
     MsmAccessRight[] result = new
MsmAccessRight[reply.xdrinInt()];
     for (int i = 0; i < result.length; i++) {
         result[i] = new MsmAccessRight(reply);
     }
     reply.done();
     return result;
    }

    /**
     * Sets persistence for player.
     * @param prstp A MsmPersistence containing the persistence
to be set.
     * @exception IOException If an error has occurred.
     */
    public void setPersistence(MsmPersistence prst) throws
IOException {
     XdrBlock call = session.newCall(PLAYER_SETPERSISTENCE);
     this.xdrout(call);
     prst.xdrout(call);
     session.rpc(call).done();
    }

    /**
     * Get persistence information for player.
     * @exception IOException If an error has occurred.
     */
    public MsmPersistence getPersistence() throws IOException {
     XdrBlock call = session.newCall(PLAYER_GETPERSISTENCE);
     this.xdrout(call);
     XdrBlock reply = session.rpc(call);
     MsmPersistence result = new MsmPersistence(reply);
     reply.done();
     return result;
    }

    /**
     * Replaces a portion of the playlist for this player. The
portion to be
     * replaced and the new titles to inserted are indicated via
MsmPlaylist
     * struct pointed to by playlistp.
     * @param playlist A MsmPlaylist that indicates the period on
```

```
the playlist
     *   to be (re)scheduled and the new titles to place within
that period.
     * @exception IOException If an error has occurred.
     */
    public void setPlaylist(MsmPlaylist playlist) throws
IOException {
     XdrBlock call = session.newCall(PLAYER_SETPLAYLIST);
     this.xdrout(call);
     playlist.xdrout(call);
     session.rpc(call).done();
    }

    /**
     * Obtains a portion of the playlist for this player.
     * @param startPosition The position within the playlist at
which to start
     *     returning status.
     * @param playlistDuration The number of milliseseconds of
the playlist for
     *   which to return status.
     * @exception IOException If an error has occurred.
     */
    public MsmPlaylist getPlaylist(long startPosition, long
playlistDuration)
     throws IOException {
         XdrBlock call = session.newCall(PLAYER_GETPLAYLIST);
         this.xdrout(call);
         call.xdroutMsmTime(startPosition);
         call.xdroutMsmTime(playlistDuration);
         XdrBlock reply = session.rpc(call);
         MsmPlaylist result = new MsmPlaylist(reply);
         reply.done();
         return result;
     }

    /**
     * Obtains the playlist for this player.
     * @exception IOException If an error has occurred.
     */
    public MsmPlaylist getPlaylist() throws IOException {
     return getPlaylist(TIME_ZERO, TIME_MAXTIME);
    }

    /**
     * MsmConnects a player to the specified destination address.
     * An error is return if play is in progress at the time of a
```

```
setConnect().
     * @param connect A MsmConnect instance containing a
transport-independent
     *   address string for the destination of Media Server data
controlled
     *   by this player.  A connectp of NULL disconnects the
player from the
     *   current destination.
     * @exception IOException If an error has occurred.
     */
    public void setConnect(MsmConnect connect) throws IOException
{
     XdrBlock call = session.newCall(PLAYER_SETCONNECT);
     this.xdrout(call);
     connect.xdrout(call);
     session.rpc(call).done();
    }

    /**
     * Get current connection for player.
     * @exception IOException If an error has occurred.
     */
    public MsmConnect getConnect() throws IOException {
     XdrBlock call = session.newCall(PLAYER_GETCONNECT);
     this.xdrout(call);
     XdrBlock reply = session.rpc(call);
     MsmConnect result = new MsmConnect(reply);
     reply.done();
     return result;
    }

    /**
     * Schedules play to commence at startDate.  Play
     * will begin at playlist startPosition and continue for
playDuration NPT
     * seconds or until paused.  An error is returned if the
player is not
     * connected.
     * Only one play() command can be pending, a second play()
overrides any
     * pending play().
     * @param speed The speed at which to play.
     * @param startPosition The position within the playlist at
which to begin
     *   play.  TIME_CURRENT means the current play position.
     * @param playDuration The duration of play.
     *   TIME_MAXTIME indicates "forever".
```

```
     * @param startDate The wall-clock time of day at which to
begin play.
     *   A value of TIME_CURRENT means start play immediately.
     * @exception IOException If an error has occurred.
     */
    public void play(
     int speed, long startPosition, long playDuration, long
startDate)
     throws IOException {
         XdrBlock call = session.newCall(PLAYER_PLAY);
         this.xdrout(call);
         call.xdroutInt(speed);
         call.xdroutMsmTime(startPosition);
         call.xdroutMsmTime(playDuration);
         call.xdroutMsmTime(startDate);
         session.rpc(call).done();
     }

    /**
     * Pauses play on the player.
     * Only one pause() command can be pending, a second pause()
     * overrides any pending pause().
     * @param pausePosition The position within the playlist at
which to pause
     *   playing.  If current play position is later than
pausePosition
     *   (taking into account the direction of play), play pauses
immediately.
     *   A value of TIME_CURRENT means stop immediately.
     * @return The time at which play actually paused.
     * @exception IOException If an error has occurred.
     */
    public long pause(long pausePosition) throws Exception {
     XdrBlock call = session.newCall(PLAYER_PAUSE);
     this.xdrout(call);
     call.xdroutMsmTime(pausePosition);
     XdrBlock reply = session.rpc(call);
     long result = reply.xdrinMsmTime();
     reply.done();
     return result;
    }

    /**
     * Resumes playing.  Play will continue until paused
     * or the end of the playlist (looped playlists play
forever).
     * @param speed The speed at which to resume play.
```

```
     * @param startPosition The position within the playlist at
which to
     *   resume play.  TIME_CURRENT means the current play
position.
     * @exception IOException If an error has occurred.
     */
    public void resume(int speed, long startPosition) throws
IOException {
     XdrBlock call = session.newCall(PLAYER_RESUME);
     this.xdrout(call);
     call.xdroutInt(speed);
     call.xdroutMsmTime(startPosition);
     session.rpc(call).done();
    }

    /**
     * Get play state for a player.
     * @return  A MsmPlayStatus instance.
     * @exception IOException If an error has occurred.
     */
    public MsmPlayStatus getPlayStatus() throws IOException {
     XdrBlock call = session.newCall(PLAYER_GETPLAYSTATUS);
     this.xdrout(call);
     XdrBlock reply = session.rpc(call);
     MsmPlayStatus result = new MsmPlayStatus(reply);
     reply.done();
     return result;
    }

    public String toString() {
     return MsmToString.playerToString(this);
    }

    private static final int HANDLELEN = 12;

    public static final long TIME_BADTIME =                    -1L;
    public static final long TIME_CURRENT =                    -2L;
    public static final long TIME_ZERO    =                     0L;
    public static final long TIME_MAXTIME = 21474836479999999999L;
    public static final long TIME_MINTIME =                     1L;

    public static final int SPEED_SCENE_REVERSE = 0;
    public static final int SPEED_FASTEST_REVERSE = 1;
    public static final int SPEED_FASTER_REVERSE = 2;
    public static final int SPEED_FAST_REVERSE = 3;
    public static final int SPEED_REVERSE = 4;
    public static final int SPEED_SLOW_REVERSE = 5;
```

```
    public static final int SPEED_SLOWER_REVERSE = 6;
    public static final int SPEED_SLOWEST_REVERSE = 7;
    public static final int SPEED_SLOWEST_FORWARD = 8;
    public static final int SPEED_SLOWER_FORWARD = 9;
    public static final int SPEED_SLOW_FORWARD = 10;
    public static final int SPEED_FORWARD = 11;
    public static final int SPEED_FAST_FORWARD = 12;
    public static final int SPEED_FASTER_FORWARD = 13;
    public static final int SPEED_FASTEST_FORWARD = 14;
    public static final int SPEED_SCENE_FORWARD = 15;

    private static final int PROG = 0x206d736d;
    private static final int VERS = 1;

    private static final int SERVER_AUTHTYPE       =  1;
    private static final int PLAYER_CREATE         =  2;
    private static final int PLAYER_DELETE         =  3;
    private static final int PLAYER_LIST           =  4;
    private static final int PLAYER_SETACCESS      =  5;
    private static final int PLAYER_GETACCESS      =  6;
    private static final int PLAYER_SETPERSISTENCE =  7;
    private static final int PLAYER_GETPERSISTENCE =  8;
    private static final int PLAYER_SETPLAYLIST    =  9;
    private static final int PLAYER_GETPLAYLIST    = 10;
    private static final int PLAYER_SETCONNECT     = 11;
    private static final int PLAYER_GETCONNECT     = 12;
    private static final int PLAYER_PLAY           = 13;
    private static final int PLAYER_PAUSE          = 14;
    private static final int PLAYER_RESUME         = 15;
    private static final int PLAYER_GETPLAYSTATUS  = 16;
    private static final int TITLE_GETSTATUS       = 17;
}
```

MsmSession

```
/*
 * @(#)MsmSession.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.io.*;
import java.net.*;
import java.util.*;

/**
 * Media Stream Manager Client API
 *
 * The Media Stream Manager (msm) API provides an RPC interface
for managing
 * the scheduling and play of isochronous media streams.
 */
public class MsmSession {
    private String serverHostName;
    private Socket socket;
    private InputStream is;
    private OutputStream os;
    private int prog;
    private int vers;

    /**
     * Create a RPC session for the named server.
     * @param serverHostName The host name of a MSM server.
     * @exception IOException If an error has occurred.
     */
    public MsmSession(String serverHostName) throws IOException {
     this.serverHostName = serverHostName;
     socket = new Socket(serverHostName, pmapGetPort());
     is = new BufferedInputStream(socket.getInputStream());
     os = new BufferedOutputStream(socket.getOutputStream());
    }

    private int pmapGetPort() throws IOException {
```

```
        PortMapper pmap = null;
        try {
            pmap = new PortMapper(serverHostName);
            int port;
            prog = 100236;
            vers = 1;
            port = pmap.getPort(prog, vers, PortMapper.IPPROTO_TCP);
            if (port != 0) return port;
            prog = 0x206d736d;
            vers = 1;
            port = pmap.getPort(prog, vers, PortMapper.IPPROTO_TCP);
            if (port != 0) return port;
        } finally {
            if (pmap != null) pmap.close();
        }
        throw new MsmException("no msm server on "+serverHostName);
    }

    /**
     * Closes a session with an MSM server.
     * @exception MsmException If an error has occurred.
     */
    public void close() throws IOException {
        socket.close();
    }

    /**
     * All players on this server.
     * @return an array of all players.
     * @exception IOException If an error has occurred.
     */
    public MsmPlayer[] players() throws IOException {
        XdrBlock reply = rpc(newCall(PLAYER_LIST));
        MsmPlayer[] result = new MsmPlayer[reply.xdrinInt()];
        for (int i = 0; i < result.length; i++) {
            result[i] = new MsmPlayer(this, reply);
        }
        reply.done();
        return result;
    }

    /**
     * Obtains status about titles.
     * @param titleName The name of the title on which to obtain
status.
     * @return the status of the title.
     * @exception IOException If an error has occurred.
```

```
     */
    public MsmTitle getTitleStatus(String titleName) throws
IOException {
     XdrBlock call = newCall(TITLE_GETSTATUS);
     call.xdroutString(titleName);
     XdrBlock reply = rpc(call);
        MsmTitle result = new MsmTitle(reply);
     reply.done();
     return result;
    }

    /**
     * Returns the server host name.
     */
    public String getServerHostName() {
     return serverHostName;
    }

    XdrBlock newCall(int proc) {
     return new XdrBlock(prog, vers, proc);
    }

    synchronized XdrBlock rpc(XdrBlock call) throws IOException {
     call.send(os);
     XdrBlock reply = new XdrBlock(is);
     try {
         reply.xdrinReplyHeader(call.callXid());
     } catch (IOException e) {
         throw new MsmException(call.callProc(), e.getMessage());
     }
     int err = reply.xdrinInt();
     if (err != 0) throw new MsmException(call.callProc(), err);
     return reply;
    }

    public String toString() {
     return MsmToString.sessionToString(this);
    }

    private static final int SERVER_AUTHTYPE       =  1;
    private static final int PLAYER_CREATE         =  2;
    private static final int PLAYER_DELETE         =  3;
    private static final int PLAYER_LIST           =  4;
    private static final int PLAYER_SETACCESS      =  5;
    private static final int PLAYER_GETACCESS      =  6;
    private static final int PLAYER_SETPERSISTENCE =  7;
    private static final int PLAYER_GETPERSISTENCE =  8;
```

```
    private static final int PLAYER_SETPLAYLIST    =  9;
    private static final int PLAYER_GETPLAYLIST    = 10;
    private static final int PLAYER_SETCONNECT     = 11;
    private static final int PLAYER_GETCONNECT     = 12;
    private static final int PLAYER_PLAY           = 13;
    private static final int PLAYER_PAUSE          = 14;
    private static final int PLAYER_RESUME         = 15;
    private static final int PLAYER_GETPLAYSTATUS  = 16;
    private static final int TITLE_GETSTATUS       = 17;
}
```

MsmAccessRight

```
/*
 * @(#)MsmAccessRight.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

/**
 * Access types, operations on access lists, and rights and
 * lists of access rights.
 * Access types (read, admin, control) are the access catagories
 * defined by the MSM server (see MSM doc for each request to
 * determine the access catagory of that request).  Access op's
 * are the operations that can be made to alter access rights of
 * a particular user.  An access right is the pairing of access
 * catagories with a particular user.  An access list is a
collection
 * of access rights for multiple users.
 */
public class MsmAccessRight {
    public String name;
    public int access;
    public int op;

    public MsmAccessRight(String name, int access, int op) {
     this.name = name;
     this.access = access;
     this.op = op;
    }

    MsmAccessRight(XdrBlock xdr) {
     name = xdr.xdrinString();
     access = xdr.xdrinInt();
     op = xdr.xdrinInt();
    }

    void xdrout(XdrBlock xdr) {
     xdr.xdroutString(name);
     xdr.xdroutInt(access);
```

```
        xdr.xdroutInt(op);
    }

    public String toString() {
        return MsmToString.accessRightToString(this);
    }

    public static final int ACCESS_NONE = 0;
    public static final int ACCESS_ADMIN = 1;
    public static final int ACCESS_READ = 2;
    public static final int ACCESS_CONTROL = 4;
    public static final int ACCESS_ALL = 7;

    public static final int OP_ADD = 0;
    public static final int OP_REMOVE = 1;
}
```

MsmPersistence

```
/*
 * @(#)MsmPersistence.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

/**
 * MsmPersistence information
 */
public class MsmPersistence {
    /**
     * Indicates the date at which the player should be
     * automatically deleted.  On terminateDate, play if in
progress, will
     * be stopped and the player deleted.  A terminateDate of
MSMTIME_MAXTIME
     * indicates the player should never be automatically
deleted.
     */
    public long terminateDate;

    public int type;

    public MsmPersistence(int type, long terminateDate) {
     this.type = type;
     this.terminateDate = terminateDate;
    }

    MsmPersistence(XdrBlock xdr) {
     type = xdr.xdrinInt();
     terminateDate = xdr.xdrinMsmTime();
    }

    void xdrout(XdrBlock xdr) {
     xdr.xdroutInt(type);
     xdr.xdroutMsmTime(terminateDate);
    }
```

```
        public String toString() {
         return MsmToString.persistenceToString(this);
        }

        /**
         * No persistence across server outage.
         */
        public static final int TYPE_NONE = 0;
        /**
         * Only public static state is preserved, play not is not
    restarted.
         */
        public static final int TYPE_PLAYLIST = 1;
        /**
         * Play is restarted after outage at last known playPosition.
         */
        public static final int TYPE_PLAYPOSITION = 2;
        /**
         * Play is restarted after outage as appropriate for current
    date.
         */
        public static final int TYPE_PLAYCURDATE = 3;
    }
```

MsmPlaylist

```
/*
 * @(#)MsmPlaylist.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

/**
 * MsmPlaylist positions are measured in seconds and nanoseconds,
titles on a
 * playlist may be scheduled to start at any non-negative
position. (In some
 * cases it may be convenient to base playlists positions at 0;
in other
 * cases it may be better to base them with the OS representation
of
 * time-of-day.) The playlist maintains a contiguous sequence of
titles and
 * "dead air". A schedule may be edited by replacing any
contiguous
 * sub-sequence of the schedule with another sequence.  It is
also possible
 * to change the starting position of the scheduled list of
titles.  Because
 * of mfs "admission delays", title start times may slip; msm
optionally
 * allows a title to be padded with dead air that can absorb the
slip, or on
 * a slip the same title or a later title can be marked to be
truncated or a
 * later title may be "joined-in-progress" to absorb the slip and
maintain
 * schedule correspondence with clock time.
 */
public class MsmPlaylist {
    /**
     * On Get, the current modification status stamp.  On Put,
modstamp on
     * which mods are based, if modification status has changed.
```

```
Mods are
     * aborted unless modstamp == MsmPlayer.TIME_CURRENT, in
which case mods
     * are always done.
     */
    public long modstamp;

    /**
     * On Get, the starting playlist position for the returned
playlist items
     * on Put, the playlist position where items are to be
replaced.
     */
    public long editStartPosition;

    /**
     * On Get, the total duration of the items returned.  On Put,
the duration
     * of the existing playlist that is to be replaced with new
items.
     *
     * NOTE: On Put, edit range specified by editStartPosition
for length
     * editDuration must lie entirely within existing playlist.
Use
     * MsmPlayer.getPlaylist() to get listStartPosition and
listDuration to
     * determine playlist bounds.
     */
    public long editDuration;

    /**
     * On Get, the startPosition for the entire playlist.  On
Put, the new
     * startPosition for the playlist after edits.
     */
    public long listStartPosition;

    /**
     * On Get, the duration of the entire list.  On Put, ignored.
     */
    public long listDuration;

    public MsmItem[] items;

    /**
     * On Get, the current loop state of the playlist.  On Put,
```

```
if TRUE, the
     * playlist wraps from end->start, start-end.
     */
    public boolean isLoop;

    public MsmPlaylist(long modstamp, boolean isLoop, long
editStartPosition,
                long editDuration, MsmItem[] items,
                long listStartPosition, long listDuration) {
     this.modstamp = modstamp;
     this.isLoop = isLoop;
     this.editStartPosition = editStartPosition;
     this.editDuration = editDuration;
     this.items = items;
     this.listStartPosition = listStartPosition;
     this.listDuration = listDuration;
    }

    MsmPlaylist(XdrBlock xdr) {
     modstamp = xdr.xdrinMsmTime();
     isLoop = xdr.xdrinBoolean();
     editStartPosition = xdr.xdrinMsmTime();
     editDuration = xdr.xdrinMsmTime();
     items = new MsmItem[xdr.xdrinInt()];
     for (int i = 0; i < items.length; i++) {
         int itemType = xdr.xdrinInt();
         switch (itemType) {
         case TITLE:
          items[i] = new MsmTitleItem(xdr);
          break;
         case DEADAIR:
          items[i] = new MsmDeadAirItem(xdr);
          break;
         }
     }
     listStartPosition = xdr.xdrinMsmTime();
     listDuration = xdr.xdrinMsmTime();
    }

    void xdrout(XdrBlock xdr) {
     xdr.xdroutMsmTime(modstamp);
     xdr.xdroutBoolean(isLoop);
     xdr.xdroutMsmTime(editStartPosition);
     xdr.xdroutMsmTime(editDuration);
     xdr.xdroutInt(items.length);
     for (int i = 0; i < items.length; i++) {
         if (items[i] instanceof MsmTitleItem) {
```

```
                xdr.xdroutInt(TITLE);
                ((MsmTitleItem)items[i]).xdrout(xdr);
               } else {
                xdr.xdroutInt(DEADAIR);
                ((MsmDeadAirItem)items[i]).xdrout(xdr);
               }
              }
              xdr.xdroutMsmTime(listStartPosition);
              xdr.xdroutMsmTime(listDuration);
             }

             public String toString() {
              return MsmToString.playlistToString(this);
             }

             private static final int TITLE   = 0;
             private static final int DEADAIR = 1;

        }
```

MsmConnect

```
/*
 * @(#)MsmConnect.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

/**
 * Connection paramaters.
 * These parameters are passed directly to mfs_str_open().
 */

public class MsmConnect {
    /**
     * The transport independent address.
     **/
    public String destTiAddr;

    /**
     * The packet encapsulation specifier (eg. MPEG Transport, *
DSS, etc).
     */
    public String encap;

    /**
     * The bits/second network bandwidth to request.
     */
    public int rate;

    public MsmConnect(String destTiAddr, String encap, int rate)
{
     this.destTiAddr = destTiAddr;
     this.encap = encap;
     this.rate = rate;
    }

    MsmConnect(XdrBlock xdr) {
     destTiAddr = xdr.xdrinString();
     encap = xdr.xdrinString();
```

```
        rate = xdr.xdrinInt();
       }

       void xdrout(XdrBlock xdr) {
        xdr.xdroutString(destTiAddr);
        xdr.xdroutString(encap);
        xdr.xdroutInt(rate);
       }

       public String toString() {
        return MsmToString.connectToString(this);
       }

}
```

MsmPlayStatus

```
/*
 * @(#)MsmPlayStatus.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

/**
 * MsmPlayStatus indicates the current state of the player.
 * STATE_WAIT indicates that a play command has been given, but
 * that startDate has not arrived.
 */
public class MsmPlayStatus {
    public long pausePosition;
    public long currentDate;
    public long currentPosition;
    public String info;
    public int currentState;
    public int currentSpeed;
    public boolean pausePending;

    MsmPlayStatus(XdrBlock xdr) {
     info = xdr.xdrinString();
     pausePending = xdr.xdrinBoolean();
     pausePosition = xdr.xdrinMsmTime();
     currentState = xdr.xdrinInt();
     currentSpeed = xdr.xdrinInt();
     currentDate = xdr.xdrinMsmTime();
     currentPosition = xdr.xdrinMsmTime();
    }

    public String toString() {
     return MsmToString.playStatusToString(this);
    }

    public static final int STATE_STOP = 0;
    public static final int STATE_WAIT = 1;
    public static final int STATE_PLAY = 2;

     }
```

MsmToString

```
/*
 * @(#)MsmToString.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.util.*;

class MsmToString {
    static String sessionToString(MsmSession se) {
     return "MsmSession"
         + "[serverHostName=" + se.getServerHostName()
         + "]";
    }

    static String playerToString(MsmPlayer pl) {
     byte[] h = pl.getHandle();
     StringBuffer sb = new StringBuffer(h.length*2);
     for (int i = 0; i < h.length; i++) {
         byte b = h[i];
         sb.append(Character.forDigit((b >> 4) & 0xf, 16));
         sb.append(Character.forDigit( b       & 0xf, 16));
     }
     return "MsmPlayer"
         + "[serverHostName=" +
pl.getSession().getServerHostName()
         + " handle=" + sb.toString()
         + "]";
    }

    private static final String[] rights =
{"admin","read","control"};

    private static final String[] ops = {"add","remove"};

    static String accessRightToString(MsmAccessRight ar) {
     StringBuffer sb = new StringBuffer();
     for (int i = 0; i < rights.length; i++) {
```

```
            if ((ar.access & (1 << i)) != 0) {
             if (sb.length() > 0) sb.append("|");
             sb.append(rights[i]);
            }
        }
        if (sb.length() == 0) sb.append("none");
        String op;
        if (ar.op >= 0 && ar.op < ops.length) op = ops[ar.op];
        else op = String.valueOf(ar.op);
        return "MsmAccessRight"
            + "[name=" + ar.name
               + " access=" + sb.toString()
               + " op=" + op
            + "]";
    }

    static String connectToString(MsmConnect co) {
        return "MsmConnect"
            + "[destTiAddr=\"" + co.destTiAddr +"\""
            + " encap=\"" + co.encap +"\""
               + " rate=" + co.rate
            + "]";
    }

    static String deadAirItemToString(MsmDeadAirItem dai) {
        return "MsmDeadAirItem"
            + "[itemDuration=" + dai.itemDuration
               + " joinInDuration=" + dai.joinInDuration
            + "]";
    }

    private static final String[] types = {
        "none","playlist","playposition","playcurdate"};

    static String persistenceToString(MsmPersistence pe) {
        String type;
        if (pe.type >= 0 && pe.type < types.length) type =
types[pe.type];
        else type = String.valueOf(pe.type);
        return "MsmPersistence"
            + "[type=" + type
               + "
terminateDate=\""+dateToString(pe.terminateDate)+"\""
            + "]";
    }

    static String dateToString(long date) {
```

```
        if (date == MsmPlayer.TIME_MAXTIME) return "never";
        else return new Date(date/1000000L).toString();
      }

    private static final String[] states =
{"stop","wait","play"};

    private static final String[] speeds = {
      "scene_reverse","fastest_reverse","faster_reverse","fast_rev
erse",
      "reverse","slow_reverse","slower_reverse","slowest_reverse",
      "slowest_forward","slower_forward","slow_forward","forward",
      "fast_forward","faster_forward","fastest_forward","scene_for
ward"};

    static String playStatusToString(MsmPlayStatus ps) {
      String state;
      if (ps.currentState >= 0 && ps.currentState < states.length)
{
          state = states[ps.currentState];
      } else state = String.valueOf(ps.currentState);
      String speed;
      if (ps.currentSpeed >= 0 && ps.currentSpeed < speeds.length)
{
          speed = speeds[ps.currentSpeed];
      } else speed = String.valueOf(ps.currentSpeed);
      return "MsmPlayStatus"
          + "[info=\"" + ps.info +"\""
             + " pausePending=" + ps.pausePending
             + " pausePosition=" + ps.pausePosition
             + " currentState=" + state
             + " currentSpeed=" + speed
             + " currentDate=\"" + dateToString(ps.currentDate) +
"\""
             + " currentPosition=" + ps.currentPosition
          + "]";
    }

    static String playlistToString(MsmPlaylist pl) {
      StringBuffer sb = new StringBuffer();
      if (pl.items != null) {
          for (int i = 0; i < pl.items.length; i++) {
            if (i != 0) sb.append(",");
            sb.append(pl.items[i].toString());
          }
      }
      return "MsmPlaylist"
```

```
            + "[modstamp=\"" + dateToString(pl.modstamp) + "\""
            + " isLoop=" + pl.isLoop
            + " editStartPosition=" + pl.editStartPosition
            + " editDuration=" + pl.editDuration
            + " items=[" + sb.toString() + "]"
            + " listStartPosition=" + pl.listStartPosition
            + " listDuration=" + pl.listDuration
            + "]";
        }

        static String titleToString(MsmTitle ti) {
         StringBuffer sb = new StringBuffer();
         if (ti.speedScale != null) {
             for (int i = 0; i < ti.speedScale.length; i++) {
              if (i != 0) sb.append(",");
              sb.append(ti.speedScale[i]);
             }
         }
         return "MsmTitle"
             + "[name=\"" + ti.name + "\""
             + " speedScale=[" + sb.toString() + "]"
             + " maxBitRate=" + ti.maxBitRate
             + " totalPlayDuration=" + ti.totalPlayDuration
             + " format=\"" + ti.format + "\""
             + "]";
        }

        static String titleItemToString(MsmTitleItem ti) {
         return "MsmTitleItem"
             + "[titleName=\"" + ti.titleName + "\""
                + " itemDuration=" + ti.itemDuration
                + " startOffset=" + ti.startOffset
                + " playDuration=" + ti.playDuration
                + " joinInDuration=" + ti.joinInDuration
             + " isTimeLocked=" + ti.isTimeLocked
             + " playClosestSpeed=" + ti.playClosestSpeed
                + " maxBitRate=" + ti.maxBitRate
             + "]";
        }

    }
```

MsmItem

```
/*
 * @(#)MsmItem.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

public abstract class MsmItem {
    /**
     * The number of milliseconds allocated to this item.
     */
    public long itemDuration;

    /**
     * Time of initial play that may be sacrificed to absorb
previous schedule
     * slips.   Silently limited to itemDuration.  If
TIME_CURRENT,
     * itemDuration is used.
     */
    public long joinInDuration;
}
```

MsmTitleItem

```
/*
 * @(#)MsmTitleItem.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

/*
 * A playlist title item.
 */
public class MsmTitleItem extends MsmItem {
    /**
     * The number of milliseconds into title where play should
begin.  It is
     * illegal for this to be greater than the total play time of
the title.
     */
    public long startOffset;

    /**
     * The number of milliseconds of title to play within this
item.
     * Values less than itemDuration allow some pad for absorbing
admission
     * delays (and the play truncation that would occur), but
should admission
     * delay be zero, dead air would occur for the remainder of
the item. It
     * is illegal for playDuration to be greater than
itemDuration or for
     * playDuration + startOffset to be greater than the total
play time of
     * the title. If TIME_CURRENT, the min of itemDuration and
total play time
     * minus startOffset is used.
     */
    public long playDuration;

    /**
```

```
     * The file pathname for title.
     */
    public String titleName;

    /**
     * Ignored on MsmPlayer.setPlaylist.  Returns max bit rate of
title on
     * MsmPlayer.getPlaylist.
     */
    public int maxBitRate;

    /**
     * If true, terminate play after itemDuration seconds (even
if admission
     * delays have caused schedule to slip and title has not
completed).  If
     * false, always play itemDuration seconds of title, allow
schedule to
     * slip if necessary.
     */
    public boolean isTimeLocked;

    /**
     * If true, plays closest available speed in same direction
if requested
     * speed is not available.  Search for closest is proceeds
towards normal
     * presentation rate.  Play is skipped if normal presentation
rate in
     * direction is not available.  If false, play of title is
skipped if
     * appropriate speed is not available.
     */
    public boolean playClosestSpeed;

    public MsmTitleItem(String titleName, long itemDuration, long
startOffset,
                        long playDuration, long joinInDuration,
                        boolean isTimeLocked, boolean playClosestSpeed,
                        int maxBitRate) {
        this.titleName = titleName;
        this.itemDuration = itemDuration;
        this.startOffset = startOffset;
        this.playDuration = playDuration;
        this.joinInDuration = joinInDuration;
        this.isTimeLocked = isTimeLocked;
        this.playClosestSpeed = playClosestSpeed;
```

```
        this.maxBitRate = maxBitRate;
      }

      MsmTitleItem(XdrBlock xdr) {
        titleName = xdr.xdrinString();
        itemDuration = xdr.xdrinMsmTime();
        startOffset = xdr.xdrinMsmTime();
        playDuration = xdr.xdrinMsmTime();
        joinInDuration = xdr.xdrinMsmTime();
        isTimeLocked = xdr.xdrinBoolean();
        playClosestSpeed = xdr.xdrinBoolean();
        maxBitRate = xdr.xdrinInt();
      }

      void xdrout(XdrBlock xdr) {
        xdr.xdroutString(titleName);
        xdr.xdroutMsmTime(itemDuration);
        xdr.xdroutMsmTime(startOffset);
        xdr.xdroutMsmTime(playDuration);
        xdr.xdroutMsmTime(joinInDuration);
        xdr.xdroutBoolean(isTimeLocked);
        xdr.xdroutBoolean(playClosestSpeed);
        xdr.xdroutInt(maxBitRate);
      }

      public String toString() {
        return MsmToString.titleItemToString(this);
      }

}
```

MsmDeadAirItem

```
/*
 * @(#)MsmDeadAirItem.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

public class MsmDeadAirItem extends MsmItem {
    public MsmDeadAirItem(long itemDuration, long joinInDuration)
{
     this.itemDuration = itemDuration;
     this.joinInDuration = joinInDuration;
    }

    MsmDeadAirItem(XdrBlock xdr) {
     itemDuration = xdr.xdrinMsmTime();
     joinInDuration = xdr.xdrinMsmTime();
    }

    void xdrout(XdrBlock xdr) {
     xdr.xdroutMsmTime(itemDuration);
     xdr.xdroutMsmTime(joinInDuration);
    }

    public String toString() {
     return MsmToString.deadAirItemToString(this);
    }

}
```

MsmException

```
/*
 * @(#)MsmException.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.io.*;

/**
 * Signals that an Media Stream Manager exception has occurred.
 */
public class MsmException extends IOException {
    /**
     * Constructs an MsmException with no detail message.
     * A detail message is a String that describes this
particular exception.
     */
    MsmException() {
     super();
    }

    /**
     * Constructs an MsmException with the specified detail
message.
     * A detail message is a String that describes this
particular exception.
     * @param s the detail message
     */
    MsmException(String s) {
     super(s);
    }

    MsmException(int proc, String msg) {
     super(((proc >= 0 && proc < procNames.length) ?
            procNames[proc] : Integer.toString(proc))
           + ": " +
           msg);
    }
```

```
MsmException(int proc, int err) {
 super(((proc >= 0 && proc < procNames.length) ?
        procNames[proc] : Integer.toString(proc))
       + ": " +
       ((err >= 0 && err < errNames.length) ?
        errNames[err] : Integer.toString(err)));
}

private static final String[] procNames = {
 "null",
 "server authtype",
 "player create",
 "player delete",
 "player list",
 "player access set",
 "player access get",
 "player persistence set",
 "player persistence get",
 "player playlist set",
 "player playlist get",
 "player connect set",
 "player connect get",
 "player play",
 "player pause",
 "player resume",
 "player play status",
 "title status",
};

private static final String[] errNames = {
 "success",                /*  0 */
 "failed",                 /*  1 */
 "badarg",                 /*  2 */
 "no mem",                 /*  3 */
 "no netname",             /*  4 */
 "des auth failed",        /*  5 */
 "kerb auth failed",       /*  6 */
 "no such player",         /*  7 */
 "old modstamp",           /*  8 */
 "item overlap",           /*  9 */
 "bad speed",              /* 10 */
 "bad start date",         /* 11 */
 "not connected",          /* 12 */
 "bad pause position",     /* 13 */
 "play active",            /* 14 */
 "bad file name",     /* 15 */
 "bad mfs file",           /* 16 */
```

```
        "bad file type",     /* 17 */
        "info too long",     /* 18 */
        "auth failed",       /* 19 */
        "bad position",          /* 20 */
        "kerberos unsupported",  /* 21 */
        "bad credentials",  /* 22 */
        "insufficient authorization", /* 23 */
        "bad access op",     /* 24 */
        "bad access type",  /* 25 */
        "bad persist type", /* 26 */
        "bad time arg",          /* 27 */
        "bad start position",    /* 28 */
        "bad duration",          /* 29 */
        "bad start offset", /* 30 */
        "bad edit start pos",    /* 31 */
        "bad edit duration",     /* 32 */
        "bad list start pos",    /* 33 */
        "bad item duration",     /* 34 */
        "bad join in duration",  /* 35 */
        "bad play duration",     /* 36 */
        "bad item type",     /* 37 */
        "bad title type",    /* 38 */
        "no such file",          /* 39 */
        "bad lut file",          /* 40 */
        "bad mfs fs",        /* 41 */
        "toc syntax",        /* 42 */
        "toc eof",           /* 43 */
        "toc bad char",          /* 44 */
        "no normal speed",   /* 45 */
        "dup speeds",        /* 46 */
        "bad file len",          /* 47 */
        "toc incomplete",    /* 48 */
        "toc can't map",     /* 49 */
        "toc bad filesize", /* 50 */
        "toc bad index",     /* 51 */
        "too low connect rate",  /* 52 */
    };

}
```

XdrBlock

```
/*
 * @(#)XdrBlock.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.io.*;
import java.net.*;

/**
 * Used to manipulate ONC RPC calls and replies.
 */
class XdrBlock {
    byte[] buf;
    int ptr;

    /*
     * Create a new empty block.
     * @param size The size of the block.
     */
    public XdrBlock(int size) {
     buf = new byte[size];
    }

    /*
     * Create a new empty block.
     */
    public XdrBlock() {
     this(256);
    }

    /*
     * Create a new block and initialize it with a call header.
     * @param prog The RPC program number.
     * @param vers The RPC version number.
     * @param proc The RPC procedure number.
     * @return The xid generated.
     */
```

```
public XdrBlock(int prog, int vers, int proc) {
 this();
 xdroutCallHeader(prog, vers, proc);
}

/**
 * Create a new block and receive it from an InputStream.
 * @param is The InputStream from which to receive the block.
 * @exception IOException If an IO error has occurred.
 */
public XdrBlock(InputStream is) throws IOException {
 synchronized (is) {
     int hdr;
     do {
      hdr  = readByte(is) << 24;
      hdr |= readByte(is) << 16;
      hdr |= readByte(is) <<  8;
      hdr |= readByte(is)      ;
      int start;
      int count = hdr & 0x7fffffff;
      if (buf == null) {
          start = 0;
          buf = new byte[count];
      } else {
          start = buf.length;
          byte[] tmp = new byte[start + count];
          System.arraycopy(buf, 0, tmp, 0, start);
          buf = tmp;
      }
      while (count > 0) {
          int done = is.read(buf, start, count);
          if (done < 0) throw new IOException("end of file");
          start += done;
          count -= done;
      }
     } while ((hdr & 0x80000000) == 0);
 }
}

private int readByte(InputStream is) throws IOException {
 int result = is.read();
 if (result < 0) throw new IOException("end of file");
 return result;
}

/**
 * Send the block to an output stream.
```

```
     * @param is The OutputStream ro which to send the block.
     * @exception IOException If an IO error has occurred.
     */
    public synchronized void send(OutputStream os) throws
IOException {
     int hdr = ptr | 0x80000000;
     synchronized (os) {
         os.write((hdr >> 24) & 0xff);
         os.write((hdr >> 16) & 0xff);
         os.write((hdr >>  8) & 0xff);
         os.write((hdr      ) & 0xff);
         os.write(buf, 0, ptr);
         if (os instanceof BufferedOutputStream) {
          ((BufferedOutputStream)os).flush();
         }
     }
    }

    /**
     * Input a fixed-length array of bytes from the block.
     * @param len The lenght of the array.
     * @return The byte array.
     */
    public synchronized byte[] xdrinBytes(int len) {
     byte[] result = new byte[len];
     System.arraycopy(buf, ptr, result, 0, len);
     ptr = (ptr + len + 3) & -4;
     return result;
    }

    /**
     * Input a variable-length array of bytes from the block.
     * @return The byte array.
     */
    public synchronized byte[] xdrinBytes() {
     return xdrinBytes(xdrinInt());
    }

    /**
     * Input an int from the block.
     * @return The int.
     */
    public synchronized int xdrinInt() {
     int result;
     result  = (buf[ptr    ] & 0xff) << 24;
     result |= (buf[ptr + 1] & 0xff) << 16;
     result |= (buf[ptr + 2] & 0xff) <<  8;
```

```
        result |= (buf[ptr + 3] & 0xff);
        ptr += 4;
        return result;
    }

    /**
     * Input an boolean from the block.
     * @return The boolean.
     */
    public boolean xdrinBoolean() {
        return xdrinInt() != 0;
    }

    /**
     * Input a String from the block.
     * @return The String.
     */
    public String xdrinString() {
        return new String(xdrinBytes(), 0);
    }

    /**
     * Input a Media Stream Manager Time value
     */
    public synchronized long xdrinMsmTime() {
        long sec = xdrinInt();
        long nsec = xdrinInt();
        if (sec == nsec && sec < 0) return sec;
        return sec*1000000000L + nsec;
    }

    /**
     * Output a fixed-length array of bytes to the block.
     * @param val The array to output.
     * @param len The length of the array to output.
     */
    public synchronized void xdroutBytes(byte[] val, int len) {
        int nxt = (ptr + len + 3) & -4;
        if (nxt > buf.length) grow(nxt);
        System.arraycopy(val, 0, buf, ptr, len);
        ptr = nxt;
    }

    /**
     * Output a variable-length array of bytes to the block.
     * @param val The array to output.
     */
```

```
public synchronized void xdroutBytes(byte[] val) {
 int len = val.length;
 xdroutInt(len);
 xdroutBytes(val, len);
}

/**
 * Output an int to the block.
 * @param val The int to output.
 */
public synchronized void xdroutInt(int val) {
 int nxt = ptr + 4;
 if (nxt > buf.length) grow(nxt);
 buf[ptr    ] = (byte)((val >> 24) & 0xff);
 buf[ptr + 1] = (byte)((val >> 16) & 0xff);
 buf[ptr + 2] = (byte)((val >>  8) & 0xff);
 buf[ptr + 3] = (byte)((val      ) & 0xff);
 ptr = nxt;
}

/**
 * Output an boolean to the block.
 * @param val The boolean to output.
 */
public void xdroutBoolean(boolean val) {
 xdroutInt(val? 1:0);
}

/**
 * Output a String to the block.
 * @param val The String to output.
 */
public void xdroutString(String val) {
 int len = val.length();
 byte[] tmp = new byte[len];
 val.getBytes(0, len, tmp, 0);
 xdroutBytes(tmp);
}

/**
 * Output a Media Stream Manager Time value
 * @param val The time to output.
 */
public synchronized void xdroutMsmTime(long val) {
 if (val < 0) {
     xdroutInt((int)val);
     xdroutInt((int)val);
```

```
        } else {
            xdroutInt((int)(val/1000000000L));
            xdroutInt((int)(val%1000000000L));
        }
    }

    private void grow(int needed) {
        int len = buf.length*2;
        while (len < needed) len *= 2;
        byte[] tmp = new byte[len];
        System.arraycopy(buf, 0, tmp, 0, buf.length);
        buf = tmp;
    }

    /**
     * Output a RPC Call header to the block.
     * @param prog The RPC program number.
     * @param vers The RPC version number.
     * @param proc The RPC procedure number.
     */
    public synchronized void xdroutCallHeader(int prog, int vers,
int proc) {
        xdroutInt(genXid());
        xdroutInt(CALL);
        xdroutInt(RPCVERS);
        xdroutInt(prog);
        xdroutInt(vers);
        xdroutInt(proc);
        xdroutInt(AUTH_UNIX);
        xdroutBytes(cred());
        xdroutInt(AUTH_NULL);
        xdroutBytes(verf());
    }

    public synchronized int callXid() {
        int tmp = ptr;
        ptr = 0;
        int result = xdrinInt();
        ptr = tmp;
        return result;
    }

    public synchronized int callProc() {
        int tmp = ptr;
        ptr = 20;
        int result = xdrinInt();
        ptr = tmp;
```

```
        return result;
      }

      private static int lastXid = 0;

      private synchronized static int genXid() {
       if (lastXid != 0) lastXid += 1;
       else lastXid = (int)(Math.random() * 2147483648.0D);
       return lastXid;
      }

      private static byte[] lastCred;

      private synchronized static byte[] cred() {
       if (lastCred == null) {
           XdrBlock xdr = new XdrBlock();
           xdr.xdroutInt((int)(System.currentTimeMillis()/1000L));
           String host;
           try host = InetAddress.getLocalHost().getHostName();
           catch (UnknownHostException e) host = "???";
           xdr.xdroutString(host);
           int uid;
           try uid =
Integer.parseInt(System.getProperty("user.uid"));
           catch (NumberFormatException e) uid = 0;
           xdr.xdroutInt(uid);
           int gid;
           try gid =
Integer.parseInt(System.getProperty("user.gid"));
           catch (NumberFormatException e) gid = 0;
           xdr.xdroutInt(gid);
           xdr.xdroutInt(0);    // no gids
           lastCred = new byte[xdr.ptr];
           System.arraycopy(xdr.buf, 0, lastCred, 0, xdr.ptr);
       }
       return lastCred;
      }

      private static byte[] lastVerf;

      private synchronized static byte[] verf() {
       if (lastVerf == null) {
           lastVerf = new byte[0];
       }
       return lastVerf;
      }
```

```
    /**
     * Input a RPC reply header from the block.
     * @param xid The expected xid.
     * @exception IOException If an error has occurred.
     */
    public synchronized void xdrinReplyHeader(int xid) throws
IOException {
        int replyXid = xdrinInt();
        if (replyXid != xid) {
            throw new IOException(
             "rpc xid mismatch: " +
             "expected " + xid + " but got " + replyXid);
        }
        int msgType = xdrinInt();
        if (msgType != REPLY) {
            throw new IOException(
             "rpc msg type mismatch: " +
             " expected " + REPLY + " but got " + msgType);
        }
        int replyStat = xdrinInt();
        switch (replyStat) {
        case MSG_ACCEPTED:
            int verfType = xdrinInt();
            byte[] verf = xdrinBytes();
            int acceptStat = xdrinInt();
            switch (acceptStat) {
               case SUCCESS:
             return;
            case PROG_UNAVAIL:
             throw new IOException(
                 "rpc accepted: " +
                 "remote hasn't exported program");
            case PROG_MISMATCH:
             int low = xdrinInt();
             int high = xdrinInt();
             throw new IOException(
                 "rpc accepted: " +
                 "version mismatch low=" + low + " high=" + high);
            case PROC_UNAVAIL:
             throw new IOException(
                 "rpc accepted: " +
                 "program can't support procedure");
            case GARBAGE_ARGS:
             throw new IOException(
                 "rpc accepted: " +
                 "procedure can't decode params");
            default:
```

```
            throw new IOException(
                "rpc accepted: " +
                "unknown status: " + acceptStat);
            }
        case MSG_DENIED:
            int rejectStat = xdrinInt();
            switch (rejectStat) {
            case RPC_MISMATCH:
             int low = xdrinInt();
             int high = xdrinInt();
             throw new IOException(
                 "rpc rejected: " +
                 "version mismatch low=" + low + " high=" + high);
            case AUTH_ERROR:
             int authStat = xdrinInt();
             switch (authStat) {
             case AUTH_BADCRED:
                 throw new IOException(
                  "rpc rejected: " +
                  "remote can't authenticate caller: " +
                  "bad credentials (seal broken)");
             case AUTH_REJECTEDCRED:
                 throw new IOException(
                  "rpc rejected: " +
                  "remote can't authenticate caller: " +
                  "client must begin new session");
             case AUTH_BADVERF:
                 throw new IOException(
                  "rpc rejected: " +
                  "remote can't authenticate caller: " +
                  "bad verifier (seal broken)");
             case AUTH_REJECTEDVERF:
                 throw new IOException(
                  "rpc rejected: " +
                  "remote can't authenticate caller: " +
                  "verifier expired or replayed");
             case AUTH_TOOWEAK:
                 throw new IOException(
                  "rpc rejected: " +
                  "remote can't authenticate caller: " +
                  "rejected for security reasons");
             default:
                 throw new IOException(
                  "rpc rejected: " +
                  "remote can't authenticate caller: " +
                  "unknown status: " + authStat);
             }
```

```
            default:
             throw new IOException(
                 "rpc rejected: " +
                 "unknown status: " + rejectStat);
            }
        default:
            throw new IOException("unknown rpc reply status: " +
replyStat);
        }
    }

    /*
     * Blow up if ptr hasn't reached the end of the block.
     */
    public void done() throws IOException {
     if (ptr != buf.length) {
         throw new IOException(
          (buf.length-ptr) + " extra bytes of data remaining in
reply");
        }
    }

    /*
     * Provisions for authentication of caller to service and
vice-versa are
     * provided as a part of the RPC protocol.  The call message
has two
     * authentication fields, the credentials and verifier.  The
reply
     * message has one authentication field, the response
verifier.  The RPC
     * protocol specification defines all three fields to be the
following
     * opaque type (in the eXternal Data Representation (XDR)
language [9]):
     */
    private static final int AUTH_NULL      = 0;
    private static final int AUTH_UNIX      = 1;
    private static final int AUTH_SHORT     = 2;
    private static final int AUTH_DES       = 3;

    /*
     * RPC Message protocol version 2
     */
    private static final int RPCVERS = 2;
    private static final int CALL    = 0;
    private static final int REPLY   = 1;
```

```
    /*
     * A reply to a call message can take on two forms: The
message was
     * either accepted or rejected.
     */
    private static final int MSG_ACCEPTED = 0;
    private static final int MSG_DENIED   = 1;

    /*
     * Given that a call message was accepted, the following is
the status
     * of an attempt to call a remote procedure.
     */
    private static final int SUCCESS       = 0;
    private static final int PROG_UNAVAIL  = 1;
    private static final int PROG_MISMATCH = 2;
    private static final int PROC_UNAVAIL  = 3;
    private static final int GARBAGE_ARGS  = 4;

    /*
     * Reasons why a call message was rejected:
     */
    private static final int RPC_MISMATCH = 0;
    private static final int AUTH_ERROR = 1;

    /*
     * Why authentication failed:
     */
    private static final int AUTH_BADCRED      = 1;
    private static final int AUTH_REJECTEDCRED = 2;
    private static final int AUTH_BADVERF      = 3;
    private static final int AUTH_REJECTEDVERF = 4;
    private static final int AUTH_TOOWEAK      = 5;
}
```

PortMapper

```
/*
 * @(#)PortMapper.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.io.*;
import java.net.*;

/**
 * Interface to the ONC port mapper.
 */
class PortMapper {
    private Socket socket;
    private InputStream is;
    private OutputStream os;

    /**
     * Create a port mapper client.
     * @param host The server for which we want to know the port
mappings.
     * @exception IOException If there is an error.
     */
    public PortMapper(String host) throws IOException {
     socket = new Socket(host, PMAP_PORT);
     is = new BufferedInputStream(socket.getInputStream());
     os = new BufferedOutputStream(socket.getOutputStream());
    }

    /**
     * Get the port number for a particular ONC service.
     * @param prog The RPC program number.
     * @param vers The RPC version number.
     * @param prot Either IPPROTO_TCP or IPPROTO_UDP.
     * @return The port number for the service.
     * @exception IOException If there is an error.
     */
    public synchronized int getPort(int prog, int vers, int prot)
```

```
        throws IOException {
            XdrBlock call = new XdrBlock();
            call.xdroutCallHeader(PMAP_PROG, PMAP_VERS,
    PMAPPROC_GETPORT);
            call.xdroutInt(prog);
            call.xdroutInt(vers);
            call.xdroutInt(prot);
            call.xdroutInt(0);
            call.send(os);
            XdrBlock reply = new XdrBlock(is);
            reply.xdrinReplyHeader(call.callXid());
            int result = reply.xdrinInt();
            reply.done();
            return result;
        }

        /**
         * Closes the port mapper.
         */
        public synchronized void close() throws IOException {
         socket.close();
        }

        static final int IPPROTO_TCP = 6;
        static final int IPPROTO_UDP = 17;

        private static final int PMAP_PROG = 100000;
        private static final int PMAP_VERS = 2;
        private static final int PMAP_PORT = 111;

        private static final int PMAPPROC_NULL    = 0;
        private static final int PMAPPROC_SET     = 1;
        private static final int PMAPPROC_UNSET   = 2;
        private static final int PMAPPROC_GETPORT = 3;
        private static final int PMAPPROC_DUMP    = 4;
        private static final int PMAPPROC_CALLIT  = 5;

    }
```

Decoder

```
/*
 * @(#)Decoder.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.awt.*;
import java.io.*;

public class Decoder extends Panel {
    private DecoderImpl impl;

    public Decoder() {
     setLayout(new BorderLayout());
    }

    public synchronized void init(String format, Image img,String
host,int port,String ATM)
     throws IOException {
         try {
          Class implClass = Class.forName(implClassName(format));
          if (impl == null || impl.getClass() != implClass) {
              removeAll();
              impl = (DecoderImpl)implClass.newInstance();
              add("Center", impl);
          }
          impl.init(format, img, host, port,ATM);
         } catch (ClassNotFoundException e) {
          throw new IOException(e.toString());
         } catch (IllegalAccessException e) {
          throw new IOException(e.toString());
         } catch (InstantiationException e) {
          throw new IOException(e.toString());
         }
     }

    public synchronized void paint(Graphics g) {
     if (impl != null) super.paint(g);
```

```
        else {
            Rectangle b = bounds();
            g.setColor(getBackground());
            g.fill3DRect(0, 0, b.width, b.height, true);
        }
    }

    public synchronized void stop() throws IOException {
        if (impl != null) impl.stop();
    }

    public synchronized void pause() throws IOException {
        if (impl != null) impl.pause();
    }

    public synchronized void play() throws IOException {
        if (impl != null) impl.play();
    }

    public synchronized void flush() throws IOException {
        if (impl != null) impl.flush();
    }

    public synchronized String destTiAddr() throws IOException {
        if (impl != null) return impl.destTiAddr();
        return "";
    }

    public synchronized String encap() throws IOException {
        if (impl != null) return impl.encap();
        return "";
    }

    /**
     * A hacky implementation factory
     */
    private static String implClassName(String format) throws
IOException {
        String osArch = System.getProperty("os.arch", "?os.arch");
        String osName = System.getProperty("os.name", "?os.name");
        String osVersion = System.getProperty("os.version",
"?os.version");
        String spec = format + " " + osArch + " " + osName + " " +
osVersion;
        if (format.equals("MPEG1SYS")) {
            if (osName.equals("Solaris") || osName.equals("SunOS"))
{
```

```
            if (osArch.equals("sparc")) {
                return "COM.Sun.isg.smcjc.MpxDecoderImpl";
            }
          }
        }
        throw new IOException("no decoder for " + spec);
      }
    }
```

DecoderImpl

```
/*
 * @(#)DecoderImpl.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.awt.*;
import java.io.*;

abstract class DecoderImpl extends Canvas {
    public abstract void init(String format, Image img, String
host, int port,String ATM) throws IOException;
    public abstract void stop() throws IOException;
    public abstract void pause() throws IOException;
    public abstract void play() throws IOException;
    public abstract void flush() throws IOException;
    public abstract String destTiAddr() throws IOException;
    public abstract String encap() throws IOException;
}
```

MpxDecoderImpl

```
/*
 * @(#)MpxDecoderImpl.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

import java.applet.*;
import java.io.*;
import java.awt.*;
import java.net.*;

class MpxDecoderImpl extends DecoderImpl implements Runnable {
    private String format;
    private String host;
    private int port;
    private int port0;
    private Image img;
    private long fadeTimeMillis;
    private DatagramSocket ctrlSckt;
    private Thread thread;
    private DatagramPacket ctrlPckt;
    private File logFile;
    private float luminance = 1.0F;
    private int dataPort;
    private int scale = 1;
    private int state=STOP;
    private boolean multi=false;
    private boolean ATM=false;
    private String ATMs=null;

    public MpxDecoderImpl() {
     super();
    }

    public synchronized void init(String format, Image img,
String host, int port,String ATMs)
     throws IOException {
         this.format = format;
```

```
            this.img = img;
            ATM=(ATMs!=null);
               this.port=port;
               this.host=host;
               if ((port==-1)&&(!ATM)){
                dataPort = genLocalPort();
               }else{
                dataPort = port;
                port0= genLocalPort();
                   multi=!ATM;
                   if (ATM) this.ATMs = ATMs;
               }
            ctrlPckt = new DatagramPacket(
             new
byte[128],128,InetAddress.getLocalHost(),genLocalPort());
            ctrlWord(0, 0x00000001); // sync
            ctrlWord(1, 0x00000002); // sync
            ctrlWord(2, 0x00000003); // sync
            ctrlWord(3, 0x00000004); // sync
            ctrlWord(4, 0xaaaa0001); // version = 1
            ctrlWord(5, 0xbbbb0001); // channel = 1
            ctrlWord(6, 0x00000000); // sequence = 0
            ctrlWord(7,     0xcccc0000); // flags = 0
            ctrlWord(8,     0xdddd0001); // type = 1
        }

        public Dimension minimumSize() {
         return new Dimension(WIDTH, HEIGHT);
        }

        public synchronized Dimension preferredSize() {
         Dimension dim = new Dimension(WIDTH*scale, HEIGHT*scale);
         return dim;
        }

        public synchronized void layout() {
         Rectangle b = bounds();
         double xscale = (double)b.width/(double)WIDTH;
         double yscale = (double)b.height/(double)HEIGHT;
         int scale = (int)((xscale + yscale) / 2.0 + 0.25);
         if (scale < 1) scale = 1;
         if (scale > 3) scale = 3;
         if (scale != this.scale) {
             this.scale = scale;
             if (state == PAUSE || state == PLAY) updateVideoMode();
         }
        }
```

```
    public synchronized void paint(Graphics g) {
     Dimension ps = preferredSize();
     g.setColor(getBackground());
     g.fill3DRect(0, 0, ps.width, ps.height, true);
     if (img != null) g.drawImage(img, 0, 0, ps.width, ps.height,
this);
    }

    public synchronized void stop() throws IOException {
     if (state == PAUSE || state == PLAY) {

            if (multi||ATM){
              StringBuffer sc= new StringBuffer();
              sc.append("kloop ");
System.out.println(sc.toString());
              String[] cmdarray0= new String[3];
              cmdarray0[0] = "/bin/sh";
              cmdarray0[1] = "-c";
              cmdarray0[2] = sc.toString();
              try Runtime.getRuntime().exec(cmdarray0);
              catch (SecurityException e)
System.out.println("Exec="+exec(cmdarray0[2]));
            }
         ctrlWord(9,     MCMD_EXIT);
         ctrlSckt.send(ctrlPckt);
         ctrlSckt.close();
         ctrlSckt = null;
         state = STOP;
         try {
          if (logFile.length() == 0) logFile.delete();
         } catch (SecurityException e) {
          String cmd = "/bin/rm -f "+logFile.getPath();
          try Runtime.getRuntime().exec(cmd);
          catch (SecurityException f) exec(cmd);
         }
     }
    }

    public synchronized void pause() throws IOException {
     if (state == PLAY) {
         ctrlWord(9,      MCMD_PLAYCTR); // identifier
         ctrlWord(10, PC_PAUSE); // action
         ctrlWord(11, Float.floatToIntBits(1.0F)); // speed
         ctrlSckt.send(ctrlPckt);
         state = PAUSE;
     }
```

```
    }

    public synchronized void play() throws IOException {
     if (state == PAUSE) {
         ctrlWord(9,      MCMD_PLAYCTR); // identifier
         ctrlWord(10, PC_PLAY);    // action
         ctrlWord(11, Float.floatToIntBits(1.0F)); // speed
         ctrlSckt.send(ctrlPckt);
         state = PLAY;
     } else if (state == STOP) {
         StringBuffer sb = new StringBuffer();
         sb.append("exec mpx");
         if (!multi) {
               if (!ATM){
                 sb.append(" -fn udp,lp,");
               sb.append(dataPort);
                }else{
                  sb.append(" -fn udp,lp,");
               sb.append(port0);
                }
            }else{
               sb.append(" -fn udp,lp,");
            sb.append(port0);
            }
         sb.append(" -xn udp,lp,");
         sb.append(ctrlPckt.getPort());
         sb.append(" -u 2");
         sb.append(" -v ");
         int depth = getColorModel().getPixelSize();
         if (depth == 1) {
          sb.append("mono");
         } else {
          sb.append("col");
          sb.append(depth);
          if (depth == 24 && scale > 1) sb.append("B");
         }
         sb.append(",");
         sb.append(scale);
         sb.append(" -w ");
         sb.append(windowId());
         sb.append(" </dev/null");
         sb.append(" >");
System.out.println(sb.toString());
         logFile = new
File("/tmp/mpx."+System.currentTimeMillis());
         sb.append(logFile.getPath());
         sb.append(" 2>&1");
```

```
            String[] cmdarray = new String[3];
            cmdarray[0] = "/bin/sh";
            cmdarray[1] = "-c";
            cmdarray[2] = sb.toString();
            try Runtime.getRuntime().exec(cmdarray);
            catch (SecurityException e) exec(cmdarray[2]);
            ctrlSckt = new DatagramSocket();
            state = PLAY;
               if(ATM){
                 StringBuffer sc= new StringBuffer();
                 sc.append("loop a ");
                 sc.append(dataPort+" ");
                 sc.append(port0+" >sasa &");
System.out.println(sc.toString());
              String[] cmdarray0= new String[3];
              cmdarray0[0] = "/bin/sh";
              cmdarray0[1] = "-c";
              cmdarray0[2] = sc.toString();
              try Runtime.getRuntime().exec(cmdarray0);
              catch (SecurityException e)
System.out.println("Exec="+exec(cmdarray0[2]));
               }else if (multi) {
                 StringBuffer sc= new StringBuffer();
                 sc.append("loop m ");
                 sc.append(host+" ");
                 sc.append(dataPort+" ");
                 sc.append(port0+" &");
System.out.println(sc.toString());
              String[] cmdarray0= new String[3];
              cmdarray0[0] = "/bin/sh";
              cmdarray0[1] = "-c";
              cmdarray0[2] = sc.toString();
              try Runtime.getRuntime().exec(cmdarray0);
              catch (SecurityException e)
System.out.println("Exec="+exec(cmdarray0[2]));
               }
        }
      }

      public synchronized void flush() {
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
        fadeTimeMillis = System.currentTimeMillis() + 4000;
      }
```

```
    public synchronized String destTiAddr() throws
UnknownHostException {
        String phost;
     //return "be0,"+phost+","+dataPort;
     if (ATM){
          return "port=" + ATMs + ",vc=" + dataPort;
        }else {
       phost = InetAddress.getLocalHost().getHostName();
          return "host=" + phost + ",udpport=" + dataPort;
        }
    }

    public String encap() {
     return "MPEG1SYS";
    }

    private void ctrlWord(int idx, int val) {
     byte[] buf = ctrlPckt.getData();
     buf[idx*4    ] = (byte)((val >> 24) & 0xff);
     buf[idx*4 + 1] = (byte)((val >> 16) & 0xff);
     buf[idx*4 + 2] = (byte)((val >>  8) & 0xff);
     buf[idx*4 + 3] = (byte)((val      ) & 0xff);
    }

    private void updateVideoMode() {
     ctrlWord(9,  MCMD_PRESCTR); // identifier
     ctrlWord(10, PCTR_VMD|PCTR_LUM); // which
     int depth = getColorModel().getPixelSize();
     int col = (depth==1)? 0 : (depth==24&&scale>1) ? VDM_COLB :
VDM_COL;
     ctrlWord(11, (col<<8)|scale); // video mode
     ctrlWord(12, 0);          // audio mode
     ctrlWord(13, 0);          // audio_volume
     ctrlWord(14, Float.floatToIntBits(luminance)); // luminance
     ctrlWord(15, 0);          // saturation
     ctrlWord(16, 0);          // gamma
     try ctrlSckt.send(ctrlPckt); catch (IOException e);
    }

    public synchronized void run() {
     Thread currentThread = Thread.currentThread();
     try {
         while (currentThread==thread && (state==PAUSE ||
state==PLAY)) {
          long currentTimeMillis = System.currentTimeMillis();
          float last = luminance;
          if (fadeTimeMillis < currentTimeMillis) {
```

```
            if (luminance < 1.0F) luminance += 0.125F;
          } else {
            if (luminance > 0.0F) luminance -= 0.125F;
          }
          if (luminance != last) updateVideoMode();
          if (luminance >= 1.0F) return;
          try wait(125); catch (InterruptedException e);
        }
      } finally {
        if (thread == currentThread) thread = null;
      }
    }

    private int genLocalPort() throws IOException {
      DatagramSocket sckt = new DatagramSocket();
      int port = sckt.getLocalPort();
      sckt.close();
      return port;
    }

    private native int windowId();

    private native int exec(String cmd);

    protected void finalize() {
      try stop(); catch (IOException e);
    }

    private static final int WIDTH  = 352;
    private static final int HEIGHT = 240;

    private static final int STOP  = 0;
    private static final int PLAY  = 1;
    private static final int PAUSE = 2;

    /* command identifiers */
    private static final int MCMD_NULL       = 0;
    private static final int MCMD_EXIT       = 1;
    private static final int MCMD_OPENSRC    = 2;
    private static final int MCMD_CLOSESRC   = 3;
    private static final int MCMD_REENTER    = 4;
    private static final int MCMD_PLAYCTR    = 5;
    private static final int MCMD_PRESCTR    = 6;
    private static final int MCMD_STREAM     = 7;
    private static final int MCMD_SENDSTAT   = 8;
    private static final int MCMD_STATUS     = 9;
    private static final int MCMD_ACK        = 10;
```

```
/* command flags */
private static final int MCFL_SNDACK      = (1<<0);
private static final int MCFL_ORGMPX      = (1<<2);

/* command parameter values: */

/* source_type  :    MCMD_OPENSRC */
private static final int MSC_FNAME        = 1;
private static final int MSC_FDSCP        = 4;

/* flags    :    MCMD_REENTER */
private static final int MRE_FOFS         = (1<<0);
private static final int MRE_ASOPEN       = (1<<2);
private static final int MRE_STRMS        = (1<<3);
private static final int MRE_SEEKVSEQ     = (1<<4);

/* data_type    :    MCMD_OPENSRC, MCMD_REENTER */
private static final int BSTRM_11172      = (1<<0);
private static final int BSTRM_VSEQ       = (1<<1);
private static final int BSTRM_ASEQ       = (1<<2);

/* action       :    MCMD_PLAYCTR */
private static final int PC_PLAY          = (1<<0);
private static final int PC_FWDSPEED      = (1<<1);
private static final int PC_FWDSTEP       = (1<<2);
private static final int PC_PAUSE         = (1<<3);

/* which        :    MCMD_PRESCTR */
private static final int PCTR_VMD         = (1<<0);
private static final int PCTR_AMD         = (1<<1);
private static final int PCTR_AVOL        = (1<<2);
private static final int PCTR_LUM         = (1<<3);
private static final int PCTR_SAT         = (1<<4);
private static final int PCTR_GAM         = (1<<5);

/* video_mode   :    MCMD_PRESCTR
 * 0xvvzz
 *  vv : VDM_COL, VDM_COLB
 *  zz : zoom [1-3]
 */
private static final int VDM_COL          = 1;
private static final int VDM_COLB         = 2;

/* audio_mode   :    MCMD_PRESCTR
 *
 *    cccqqq
```

```
         *    ccc: channel listening selection
         *     Sxx : 1/0 -> Selection/ No Selection
         *     101 : Left
         *     110 : Right
         *     111 : Left & Right
         *    qqq: audio playback quality selection
         *     Sxx : 1/0 -> Selection/ No Selection
         *     100 : High
         *     101 : Medium
         *     110 : Low
         */

        /* stream       :    MCMD_STREAM, MCMD_OPENSRC, MCMD_REENTER

         *    vvvvvvvv.aaaaaaaa
         *    aaaaaaaa:
         *      a7: 1-> ignore stream identifier part (bits a5-a0).
         *      a6: audio stream subscription 0/ON, 1/OFF
         *      a5: 1->auto subscribe to first encountered audio
    stream,
         *        (a4-a0 = 00000).
         *    a4-a0: subscribe to a particular audio stream [0-31]
         *
         *    vvvvvvvv:
         *      v7: 1-> ignore stream identifier part, bits v5-v0
         *      v6: video stream subscription 0/ON, 1/OFF
         *      v5: 1->auto subscribe to first encountered video
    stream,
         *        (v4-v0 = 00000).
         *      v4: 0
         *    v3-v0: subscribe to particular video stream [0-15]
         *
         */

        private static final int STRM_IGNOREID   = 0x80;
        private static final int STRM_SBCOFF     = 0x40;
        private static final int STRM_AUTOSBC    = 0x20;

        static {
         try System.loadLibrary("javampx"); catch
    (UnsatisfiedLinkError e)
             System.load("/opt/SUNWsmcjc/lib/libjavampx.so");
        }
    }
```

smcrm

```
/*
 * @(#)smcrm.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

public class smcrm {
    private static byte[] parseHandle(String s) {
     int len = s.length()/2;
     byte[] h = new byte[len];
     for (int i = 0; i < len; i++) {
         h[i] = (byte) Integer.parseInt(s.substring(i*2,
(i+1)*2), 16);
     }
     return h;
    }
    public static void main (String args[]) throws Exception {
     MsmSession session = null;
     MsmPlayer player;
     if (args.length != 2) {
         System.err.println("usage: smcrm <serverName>
<playerHandle>");
         return;
     }
     try {
         session = new MsmSession(args[0]);
         player = new MsmPlayer(session, parseHandle(args[1]));
         player.delete();
     } catch (Exception e) {
         System.err.println("smcrm: " + e);
     } finally {
         if (session != null) {
          try session.close(); catch (Exception e)
              System.err.println("smcrm: " + e);
         }
     }
    }
}
```

smcstat

```
/*
 * @(#)smcstat.java
 *
 * Copyright 1995 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version     1.0
 * author Christopher Lindblad
 *
 */

package COM.Sun.isg.smcjc;

public class smcstat {
    public static void main (String args[]) throws Exception {
     MsmSession session = null;
     MsmPlayer[] players;
     if (args.length != 1) {
         System.err.println("usage: smcstat <serverName>");
         return;
     }
     try {
         session = new MsmSession(args[0]);
         players = session.players();
         System.out.println(session);
         for (int i = 0; i < players.length; i++) {
          MsmPlayer player = players[i];
          MsmPersistence persistence = player.getPersistence();
          MsmConnect connect = player.getConnect();
          MsmPlayStatus status = player.getPlayStatus();
          MsmAccessRight[] rights = player.getAccess();
          MsmPlaylist playlist = player.getPlaylist();
          System.out.println(player);
          System.out.println(persistence);
          System.out.println(connect);
          System.out.println(status);
          for (int j = 0; j < rights.length; j++) {
              System.out.println(rights[j]);
          }
          System.out.println(playlist);
          for (int j = 0; j < playlist.items.length; j++) {
              if (playlist.items[j] instanceof MsmTitleItem) {
               MsmTitleItem ti = (MsmTitleItem)playlist.items[j];
               System.out.println(
                   session.getTitleStatus(ti.titleName));
```

```
                    }
                }
            }
        } catch (Exception e) {
            System.err.println("smcstat: " + e);
        } finally {
            if (session != null) {
             try session.close(); catch (Exception e)
                System.err.println("smcstat: " + e);
            }
        }
    }
}
```

LOOP

```
/*
 * @(#)loop.c
 *
 * Copyright 1996 Sun Microsystems, Inc.  All Rights Reserved.
 *
 * version      1.0
 * author       Stephane CACHAT
 *
 */

#include <stdio.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <netdb.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>
#include <assert.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <time.h>
#include <thread.h>
#include <sys/errno.h>
#include <sys/stropts.h>
#include <fcntl.h>
#include <atm/atmioctl.h>

#ifdef TRUE
#undef TRUE
#endif

#ifdef FALSE
#undef FALSE
#endif

#define FALSE 0
#define TRUE 1
```

```
#define BUF 1024*8

  /*****************************************************
   *** Global variables                                ***
   *****************************************************/

/* Parameters */

char servername[256];
char * progName;
char *opt;
int port;
int port0;

/* Socket */

struct sockaddr_in adds;
int skt;
struct sockaddr_in addr;
struct sockaddr_in addx;
struct hostent * hp;
int len;

/* buffer */

char * buffer=NULL;

/* Multicast */

struct ip_mreq mreq;
char * host;

/* Thread */

thread_t Tpump;
int okdone=0;
int flag=1;

/* ATM */
int safd;
int ppa;
char ctlbuf[0x100];

#define vc port

  /*****************************************************
   *** Receive&transmit info Multicast                 ***
```

```
   ************************************************/

void * pumpM(void * result){
  while (flag) {                                    /*main loop*/
    len=recvfrom(skt,buffer,BUF,0,NULL,0);
    if (len) {
    sendto(skt,buffer,len,0,(struct sockaddr *)
&(addx),sizeof(addx));
    }
  }
  flag=1;
}

  /*************************************************
   *** Receive&transmit info ATM                  ***
   *************************************************/

void * pumpA(void * result){
  struct strbuf   ctl;
  struct strbuf   data;
  int             flags;
fprintf(stderr,"pumpA\n");
  ctl.buf = (char *) ctlbuf;
  ctl.maxlen = 0x100;
  ctl.len = 0;
  data.buf = (char *) buffer;
  data.maxlen = BUF;
  data.len = 0;
  flags = 0;
  while (flag) {                                    /*main loop*/
    if (getmsg(safd, &ctl, &data, &flags) < 0) {
      fprintf(stderr,"getmsg failed, errno=%d\n", errno);
      perror("");
      return;
    }
    len=data.len;
fprintf(stderr,"len=%d\n",len);
    if (len) {
    sendto(skt,buffer+4,len-4,0,(struct sockaddr *)
&(addx),sizeof(addx));
    }
  }
  flag=1;
}

  /*************************************************
   *** Collecting arguments                       ***
```

```
    ************************************************/

    void print_usage_and_exit (char* a){
        if (strlen(a)) fprintf(stderr,a);
        fprintf(stderr,"\n%s  redirect multicast or atm data stream
    to lo0\n",progName);
        fprintf(stderr,"Usage\n");
        fprintf(stderr,"%s m <Multicast address> <in port> <out
    port>\n",progName);
        fprintf(stderr,"%s a <VC> <out port>\n",progName);
        (void)exit(0);
    }

    static void collectArgs(int argc,char **argv){
        int i;
        int j=0;
        FILE * f;
        progName=*argv++;
        if (!*argv) print_usage_and_exit("");
        opt=*argv++;
        if (*opt=='a') {
          if (!*argv) print_usage_and_exit("");
          port=atoi(*argv++);
          if (!*argv) print_usage_and_exit("");
          port0=atoi(*argv++);
          if (port<=0) print_usage_and_exit("");
          if (*argv) print_usage_and_exit("");
          f=fopen("./loop.conf","r");
          if (!f){
            fprintf(stderr,"Can't open loop.conf");
            exit(-1);
          }
          host= (char*) malloc(256);
          fscanf(f,"%s",host);
          fclose(f);
        }else if (*opt=='m') {
          if (!*argv) print_usage_and_exit("");
          host=*argv++;
          if (!*argv) print_usage_and_exit("");
          port=atoi(*argv++);
          if (!*argv) print_usage_and_exit("");
          port0=atoi(*argv++);
          if (port<=0) print_usage_and_exit("");
          if (*argv) print_usage_and_exit("");
        } else print_usage_and_exit("");
    }
```

```
/*************************************************
 *** Getting server IP adress                    ***
 *************************************************/

void getaddr(){
  int udpport;
  unsigned long inaddr;
  struct hostent * hp;
  char n[256];
  int i;

  if (gethostname(servername,256)==-1)
print_usage_and_exit("error while getting hostname");
  if ((inaddr=inet_addr(servername))!=-1){
    adds.sin_addr.s_addr=inaddr;
  }else{
    hp=gethostbyname(servername);
    if (hp!=NULL){
      adds.sin_addr.s_addr=((struct in_addr*)
hp->h_addr)->s_addr;
      adds.sin_port = htons(udpport);
    }
  }
  if ((inaddr=inet_addr(host))!=-1){/*hostname*/
    mreq.imr_multiaddr.s_addr=inaddr;
  }else{
    hp=gethostbyname(host);
    if (hp!=NULL){
      mreq.imr_multiaddr.s_addr=((struct in_addr*)
hp->h_addr)->s_addr;
    }else{
      fprintf(stderr,"Multicast connect failed\n");
    }
  }
 /* mreq.imr_interface.s_addr=INADDR_ANY; */
  gethostname(n,256);
  hp=gethostbyname(n);
  if (hp!=NULL){
      mreq.imr_interface.s_addr=((struct in_addr*)
hp->h_addr)->s_addr;
      addx.sin_addr.s_addr=((struct in_addr*)
hp->h_addr)->s_addr;
      addx.sin_port = htons(port0);
  }else{
      fprintf(stderr,"Multicast connect failed\n");
  }
}
```

```
  /*****************************************************
   *** Socket setting Multicast                      ***
   *****************************************************/
void goM(){
  getaddr();
  skt=socket(AF_INET,SOCK_DGRAM,0);
  if (skt==0) {
    perror("Create socket");
    exit(EXIT_FAILURE);
  }
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(port);
  bind(skt,(void *)&addr,sizeof(addr));
  if( setsockopt(skt, IPPROTO_IP, IP_ADD_MEMBERSHIP,(char*)&mreq,
sizeof(struct ip_mreq) ) == -1 ){
    fprintf(stderr,"Can't join multicast membership");
    exit(0);
  }
  if (fcntl(skt,F_SETFL,O_NDELAY)==-1){
    fprintf(stderr,"set socket options nb");
    exit(EXIT_FAILURE);
  }

  if (thr_create(0,0,pumpM,0,0,&Tpump)) perror("Can't create
Dispatcher");
}

  /*****************************************************
   *** ATM interface setting                         ***
   *****************************************************/
void goA(){
  int udpport;
  unsigned long inaddr;
  struct hostent * hp;
  char n[256];

  char interface[10];
  memset(interface, 0, sizeof (interface));
  strcpy(interface, host);
  ppa = interface[strlen(interface) - 1] - '0';
  if ((safd = sa_open(interface)) < 0) {
    fprintf(stderr,"open failed, errno=%d\n", errno);
    perror("open");
    exit(-1);
  }
fprintf(stderr,"ready to attach\n");
```

```
  sa_attach(safd, ppa, -1);
fprintf(stderr,"attached\n");
  if (sa_add_vpci(safd, vc, NULL_ENCAP, BIG_BUF_TYPE) < 0) {
    fprintf(stderr,"sa_add_vpci failed, errno=%d\n", errno);
    exit(-1);
  }
  sa_setraw(safd);

  gethostname(n,256);
  hp=gethostbyname(n);
  if (hp!=NULL){
      addx.sin_addr.s_addr=((struct in_addr*)
hp->h_addr)->s_addr;
      addx.sin_port = htons(port0);
  }else{
      fprintf(stderr,"lo0 connect failed\n");
  }
  skt=socket(AF_INET,SOCK_DGRAM,0);
  if (skt==0) {
    perror("Create socket");
    exit(EXIT_FAILURE);
  }
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(port0);
  bind(skt,(void *)&addr,sizeof(addr));
  if (fcntl(skt,F_SETFL,O_NDELAY)==-1){
    fprintf(stderr,"set socket options nb");
    exit(EXIT_FAILURE);
  }

  if (thr_create(0,0,pumpA,0,0,&Tpump)) perror("Can't create
Dispatcher");
}

  /*************************************************
   *** Cleaning ATM                                ***
   *************************************************/

void doneA(int arg){
 fprintf(stderr,"loop killed by signal %d\n",arg);
 if (!okdone){okdone=1;
  flag=0;
  while (!flag) {
    sleep(1);
  }
  fprintf(stderr,"dispatcher killed\n");
```

```
  if (sa_delete_vpci(safd, vc) < 0) {
    fprintf(stderr,"sa_delete_vpci failed, errno=%d\n", errno);
  };
fprintf(stderr,"ready to detach\n");
  sa_detach(safd, -1);
fprintf(stderr,"detached\n");
  sa_close(safd);
  close(skt);
  printf("socket closed\n");
  if (buffer) free(buffer);
  printf("Buffer free\n");
  exit(0);
}}

  /*****************************************************
   *** Cleaning Multicast                              ***
   *****************************************************/

void doneM(int arg){
 if (!okdone){okdone=1;
  if (setsockopt(skt,IPPROTO_IP, IP_DROP_MEMBERSHIP,(char *)
&mreq,sizeof(mreq))==-1){
    fprintf(stderr,"Can't drop multicast membership");
    exit(0);
  }
  printf("Multicast membership dropped\n");

  flag=0;
  while (!flag) {
    sleep(1);
  }
  printf("dispatcher killed\n");

  close(skt);
  printf("socket closed\n");
  if (buffer) free(buffer);
  printf("Buffer free\n");
  exit(0);
}}

  /*****************************************************
   *** Main                                            ***
   *****************************************************/

int main(int argc, char** argv)
{
  int i;
```

```
    buffer=(char*) malloc(BUF);
    collectArgs(argc,argv);
  if (*opt=='m'){
     printf("host=%s, port=%d, port0=%d\n",host,port,port0);
     signal(SIGQUIT,doneM);
     signal(SIGINT,doneM);
     signal(SIGUSR1,doneM);
     signal(SIGUSR2,doneM);

     printf("go M\n");
     goM();
  }else if (*opt=='a'){
     printf("interface=%s, vc=%d,port0=%d\n",host,vc,port0);
     signal(SIGQUIT,doneA);
     signal(SIGINT,doneA);
     signal(SIGUSR1,doneA);
     signal(SIGUSR2,doneA);

     printf("go A\n");
     goA();
  }

  printf("loop\n");

  while(1) sleep(60);
 }
```

Claims

1. A method for processing in a computer which includes a memory a bit stream received from a bit stream server which is operatively coupled to the computer through a network, the method comprising:

   retrieving from a multimedia document stored in the memory a specification of a title;
   building from the specification of the title bit stream control signals which request a bit stream representing the title and which are in a form appropriate for processing by the bit stream server;
   transmitting the bit stream control signals to the bit stream server to thereby request from the bit stream server a bit stream representing the title;
   building from the specification of the title decoder control signals which direct a decoder to receive the bit stream from the bit stream server and which are in a form appropriate for processing by the decoder; and
   transmitting the decoder control signals to the decoder to thereby cause the decoder to receive and decode the bit stream.

2. An applet, capable of executing within a computer system, for requesting and controlling decoding of a bit stream specified in a multimedia document stored in a memory of the computer system, the applet comprising:

   an API module (i) which is configured to build from a specification of the bit stream in the multimedia document bit stream control signals which request transmission of the bit stream from a bit stream server and which are in a form appropriate for processing by the bit stream server and (ii) which is configured to transmit the bit stream control signals to the bit stream server to thereby request from the bit stream server a bit stream representing the title; and
   a decoder module (i) which is operatively coupled to the API module; (ii) which is configured to build from the specification of the bit stream in the multimedia document decoder control signals which direct a decoder to

receive the bit stream from the bit stream server and which are in a form appropriate for processing by the decoder; and (iii) which is configured to transmit the decoder control signals to the decoder to thereby cause the decoder to receive and decode the bit stream.

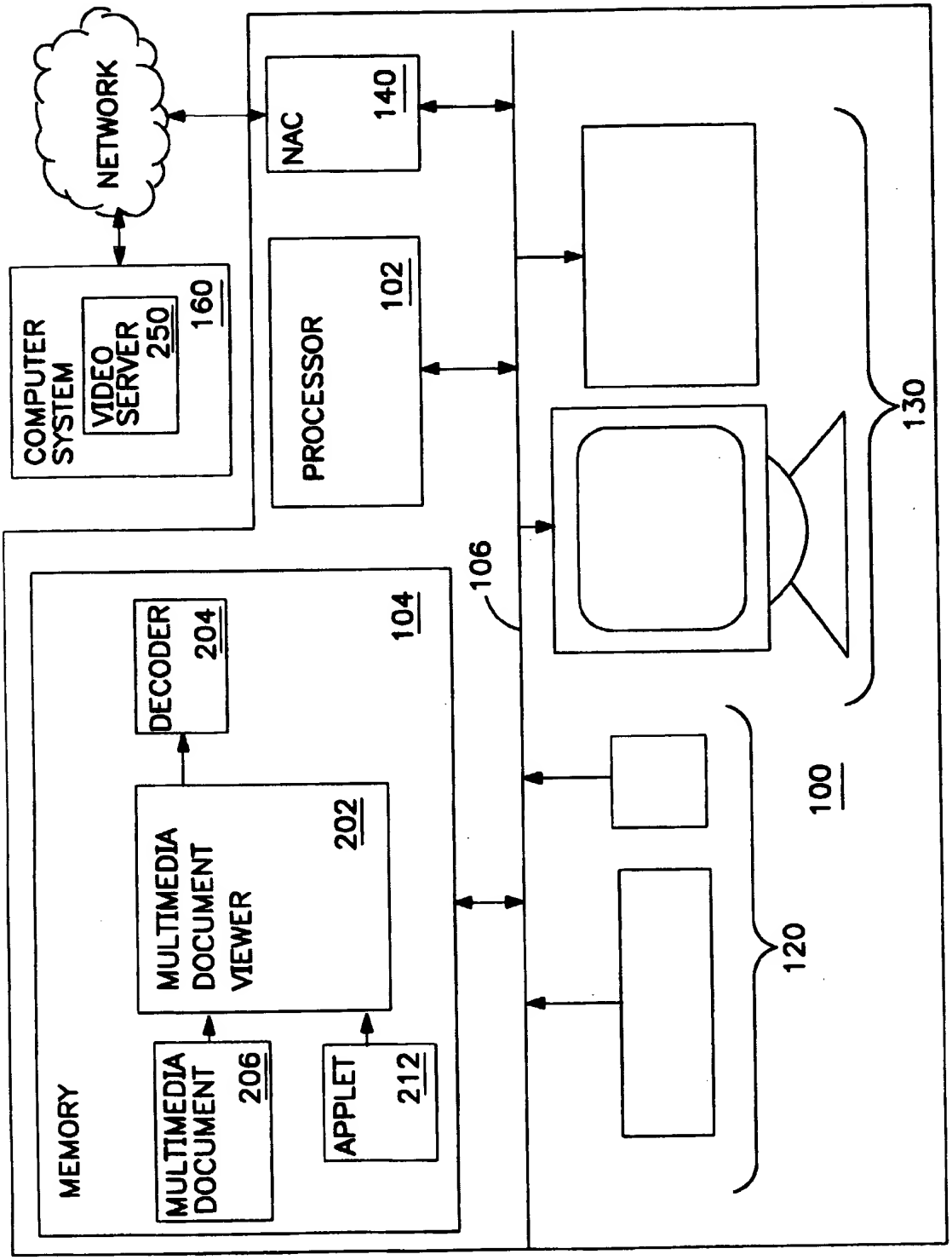
NETWORK
NAC
140
COMPUTER SYSTEM
VIDEO SERVER
250
160
PROCESSOR
102
MEMORY
DECODER
204
104
MULTIMEDIA DOCUMENT VIEWER
202
MULTIMEDIA DOCUMENT
206
APPLET
212
106
130
120
100

FIG. 1

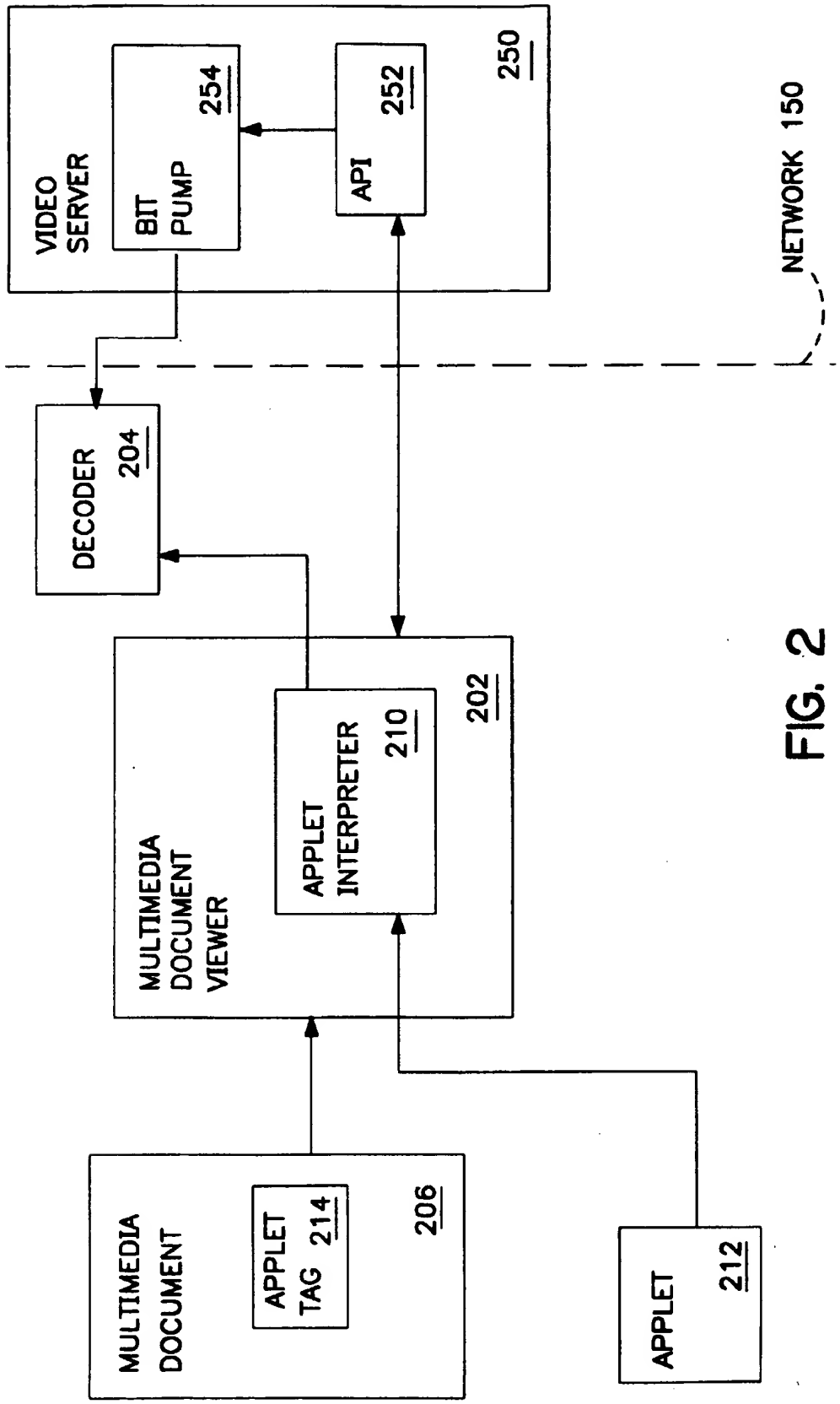
VIDEO SERVER
250
BIT PUMP
254
API
252
NETWORK 150
DECODER
204
MULTIMEDIA DOCUMENT VIEWER
202
APPLET INTERPRETER
210
MULTIMEDIA DOCUMENT
206
APPLET TAG
214
APPLET
212

FIG. 2

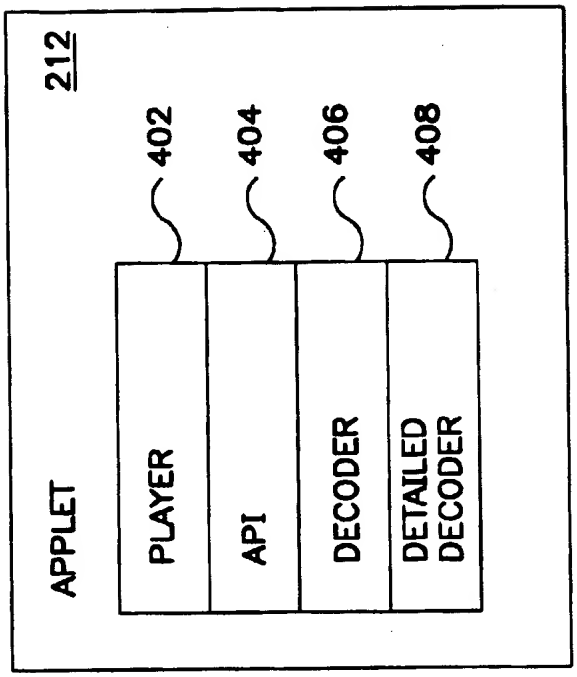
212
APPLET
PLAYER
API
DECODER
DETAILED DECODER
402
404
406
408

FIG. 4

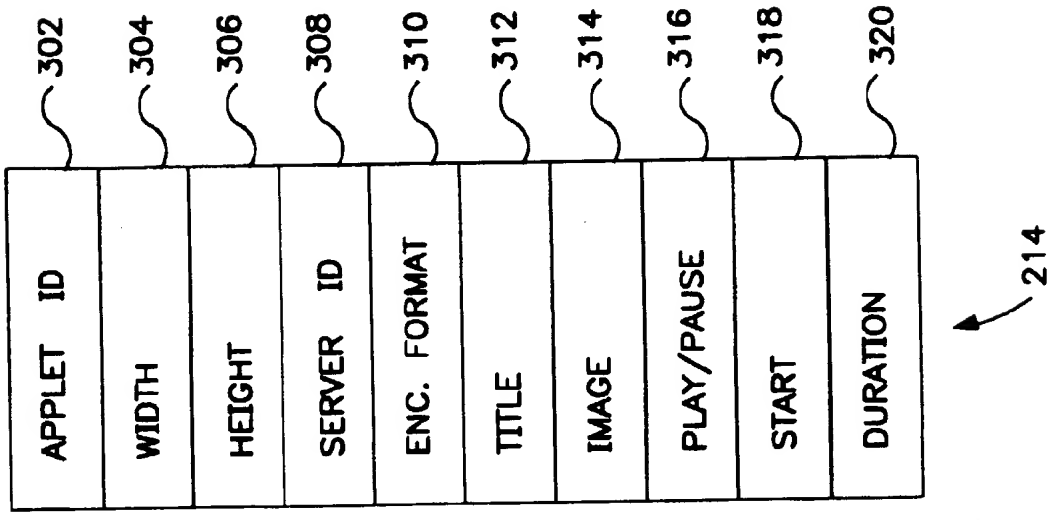
APPLET ID
WIDTH
HEIGHT
SERVER ID
ENC. FORMAT
TITLE
IMAGE
PLAY/PAUSE
START
DURATION
302
304
306
308
310
312
314
316
318
320
214

FIG. 3